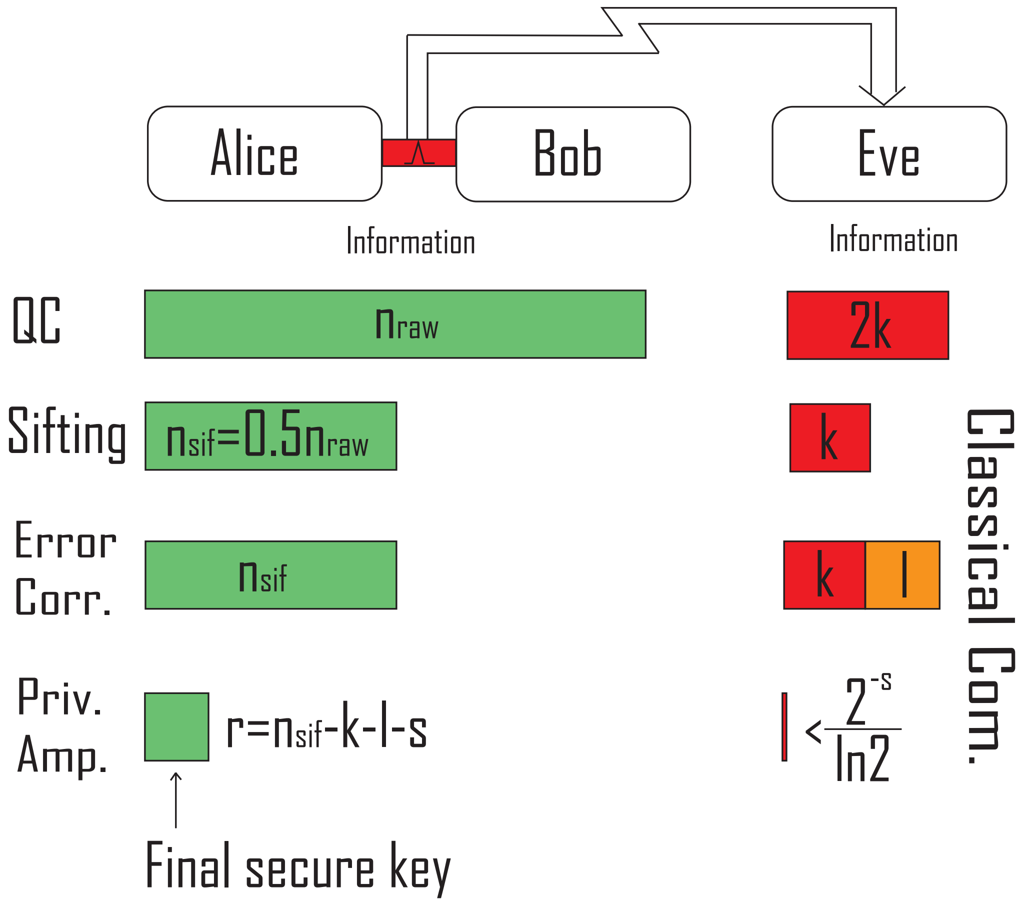
Key Reconciliation in Quantum Key Distribution

Per Grönberg



FOI-R-- 1743 --SE
ISSN 1650-1942

Sensor Technology
Technical report

October 2005

FOI
Defence Research Agency
Sensor Technology
P.O. Box 1165
SE-581 11 Linköping

Phone: +46 13 37 80 00
Fax: +46 13 37 81 00

www.foi.se

Key Reconciliation in Quantum Key Distribution

Per Grönberg

FOI-R--1743--SE
ISSN 1650-1942

Sensor Technology
Technical report

October 2005

Issuing organization FOI – Swedish Defence Research Agency Sensor Technology P.O. Box 1165 SE-581 11 Linköping	**Report number, ISRN** FOI-R--1743--SE	**Report type** Technical report
	Research area code 4. C4ISTAR	
	Month year October 2005	**Project no.** E3933
	Sub area code 41 C4I	
	Sub area code 2	
Author/s (editor/s) Per Grönberg	**Project manager** Per Jonsson	
	Approved by Ove Steinvall	
	Sponsoring agency FMV	
	Scientifically and technically responsible Per Jonsson	

Report title

Key Reconciliation in Quantum Key Distribution

Abstract

Key reconciliation in quantum key distribution is studied in this report. After the transfer of single photons on the quantum channel there will inevitably be errors due to system imperfections and possibly eavesdropping. These errors must be removed for the final key to be useful. This is done with error correction over a classical channel. A potential eavesdropper can gain information about the key both from the quantum and the classical transmission. The eavesdropper's information can be reduced to be arbitrary small in a final privacy-amplification step, where bits in the key are sacrificed to achieve the privacy.

This report presents an extensive analysis of different error-correcting codes. Four protocols for error correction have been implemented: *Cascade*, *Yamamura and Ishizuka's*, *Low Density Parity-Check Codes* and *Winnow*. Their performances have been evaluated by Monte-Carlo simulations. The number of final key bits after correction and privacy amplification, the probability of correcting all errors and the amount of information-exchange required to correct all errors have been analyzed for the different protocols. It is concluded that three of the different protocols each have characteristics that make them good choices in different setups. Only Yamamura and Ishizuka's protocol is unsuitable for a real system.

Keywords

Quantum Key Distribution, Quantum Cryptography, Quantum Information, Optical Communication, Key Reconciliation, Error Correction, Cascade, Low Density Parity-Check Codes, Winnow, Privacy Amplification

Further bibliographic information	**Language** English
ISSN 1650-1942	**Pages** 82 p.
	Price acc. to pricelist

Utgivare FOI - Totalförsvarets forskningsinstitut Sensorteknik Box 1165 581 11 Linköping	**Rapportnummer, ISRN** FOI-R--1743--SE	**Klassificering** Teknisk rapport
	Forskningsområde 4. Ledning, informationsteknik och sensorer	
	Månad, år Oktober 2005	**Projektnummer** E3933
	Delområde 41 Ledning med samband och telekom och IT-system	
	Delområde 2	
Författare/redaktör Per Grönberg	**Projektledare** Per Jonsson	
	Godkänd av Ove Steinvall	
	Uppdragsgivare/kundbeteckning FMV	
	Tekniskt och/eller vetenskapligt ansvarig Per Jonsson	

Rapportens titel

Nyckelextraktion i kvantmekanisk nyckelöverföring

Sammanfattning

Nyckelextraktion för kvantmekanisk nyckelöverföring studeras i denna rapport. Det kommer oundvikligen bli fel i transmission av enstaka fotoner på en kvantkanal, dels på grund av brister i systemet och dels på grund av eventuell tjuvlyssning. Dessa fel måste rättas för att den slutliga nyckeln ska vara användbar. Detta görs med felrättning där information överförs på en klassisk kommunikationskanal. En potentiell tjuvlyssnare kan skaffa sig information både från kvantöverföringen och från den klassiska överföringen. Tjuvlyssnarens information kan dock reduceras genom att så kallad säkerhetsförstärkning används i ett sista steg, där bitar i nyckeln offras för att den önskade säkerheten ska uppnås.

Denna rapport presenterar en omfattande analys av olika felrättningskoder. Fyra olika protokoll för felrättning har implementeras: *Cascade*, *Yamamura and Ishizuka's*, *Low Density Parity-Check Codes* och *Winnow*. Protokollens prestanda har utvärderas med Monte-Carlo simuleringar. Antal bitar i den slutliga nyckeln efter felrättning och säkerhetsförstärkning, sannolikheten att rätta alla fel och mängden information som behöver överföras på den klassiska kanalen i felrättningen har analyserats för de olika protokollen. Slutsatsen är att tre av protokollen har egenskaper som gör dem användbara i system för kvantmekanisk nyckelöverföring. Endast Yamamura and Ishizuka's protokoll är olämplig i ett verkligt system.

Nyckelord

Kvantmekanisk nyckelöverföring, Kvantkryptografi, Kvantinformation, Optisk kommunikation, Nyckelextraktion, Felrättning, Cascade, Low Density Parity-Check Codes, Winnow, Säkerhetsförstärkning

Övriga bibliografiska uppgifter	**Språk** Engelska
ISSN 1650-1942	**Antal sidor:** 82 s.
Distribution enligt missiv	**Pris: Enligt prislista**

Preface

This report is the result of a Master of Science thesis project that Per Grönberg performed at FOI. Per Grönberg will also publish a thesis at the Department of Mathematical Science at Chalmers University of Technology and Göteborg University, as a part of his degree in Master of Science in Engineering Physics with Applied Mathematics. In order not to duplicate his work of writing, this report and the thesis are nearly identical to each other. The language in this report is therefore somewhat unusual for a FOI report and resembles rather an academic thesis.

Contents

Chapter 1

Introduction

Today we live in the age of information technology. Information is easily available via the Internet to anyone who may seek it. The speed with which information travels world wide has increased astronomically in only a century. But how can I as a user trust that the information sent to me is not read by someone else on the way? This calls for the use of cryptography! In this introduction, will briefly discuss two categories of such techniques: *asymmetric* and *symmetric*.

In asymmetric cryptography the sender and receiver do not share an identical key. The most common asymmetric cryptographical algorithm is the RSA invented by Rivest, Shamir and Adleman in 1978 [1]. In RSA, each part has a private key and a public key. The public key is created from the private key. The public key is totally open for everyone and the private key is kept secret. When I want to send a message to a friend, I encrypt it using his public key and then send him the encrypted message. After the encryption the only one who can decrypt the cipher text is the one who has the matching private key, i.e. my friend. The idea is to use so-called one-way functions. Given a one-way function f and the output $y = f(x)$ it is very difficult to find the input x. The RSA is based on the factorization of large numbers into primes combined with a number of modulo operations. The private key works as a trapdoor simplifying the reversing of the one-way function. The task to turn a one-way function "inside out" is however not impossible, it is only very time consuming or "hard". In complexity theory the term "hard" corresponds to runtime exponential in the size of the problem. Furthermore it is not proved that factorization of large numbers is hard, there might be an algorithm yet to be discovered that factorizes large numbers in only polynomial time. Such an algorithm has been found for quantum computers. Today there are no large-scale quantum computers available, but it is a hot research field. In fifty years time we might have the first desktop quantum computer. Figure 1.0.1 shows the schematics of public key cryptography.

Symmetric cryptography uses a single key for both encryption and decryption. An example of a symmetric crypto is the Caesar crypto used by Caesar himself. To encrypt a message each character is replaced by the character three places ahead in the Latin alphabet. For example, *Per* would be encrypted as the cipher text *Shu*. The receiver knows how the cipher text was created, i.e. the key, and can recreate the original message. A more advanced symmetrical crypto is the DES (Data Encryption Standard) which uses a 56 bits long key as a scheme for several permutations to scramble the message. There are $2^{56} = 72,057,594,037,927,936$

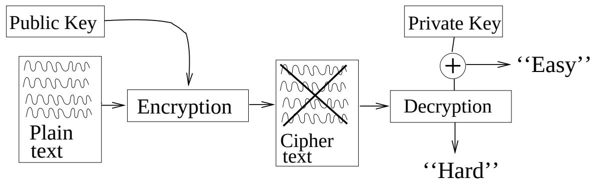


Figure 1.0.1: Schematics of public key cryptology

different combinations for the key. This is a very large number but it is still possible to find the key using an exhaustive search. An intercepted encrypted message can be stored and worked on until the correct key is eventually found. To be considered secure we need a long key and more important, we need to change the key often. Every encrypted message using a certain key also holds a clue to the key. If a key is used too often an adversary could break it using the knowledge gathered from all intercepted cipher texts. If this happens our adversary can read all encrypted messages encrypted with this key until the key is changed. Think of this as changing from three to four characters ahead in the Caesar cipher. Constantly changing this number would puzzle the enemy a lot more. This calls for the exchange of keys between two parties. This is a Catch 22, to exchange secure information we need keys which need to be exchanged securely. Today entrusted couriers are used when the demand for security is extremely high. This is expensive, slow and cumbersome. There is a need for a way to easily exchange secure keys between two points. The answer to the need could be Quantum Key Distribution, abbreviated QKD. The interested reader can find more on cryptology in the popular science book *The Code Book* by Singh [2], which also briefly touches the QKD subject.

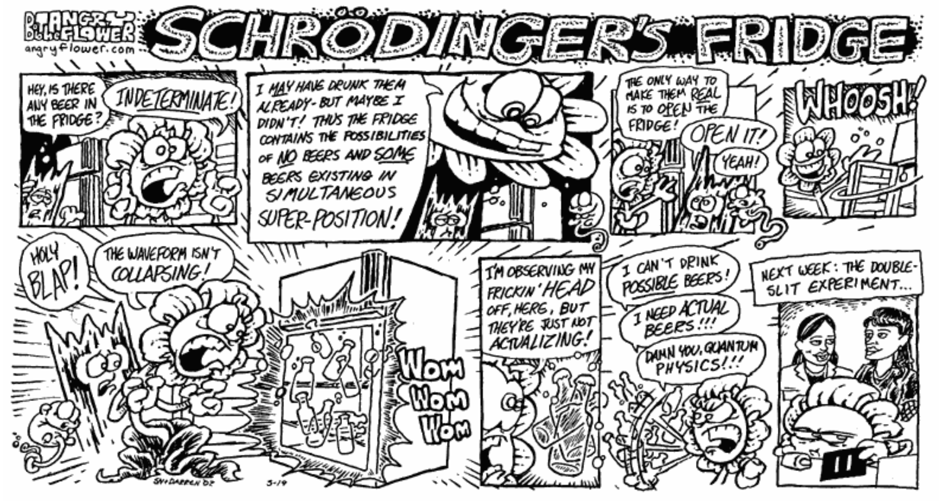
In QKD the key is transmitted from the sender, commonly called Alice, to the receiver, commonly called Bob, encoded into single photons. Due to the principles of Quantum Mechanics an adversary, commonly called Eve, can not make measurements on the single photon without disturbing its state. By disturbing the state, she will cause errors and can therefore be detected by Alice and Bob. This is the corner-stone for the proof of total security in quantum key distribution. The idea was first presented by Bennett and Brassard in 1984 [3]. Eight years later, the worlds first quantum key distribution was performed by Bennett *et al.* in 1992 [4]. The transferred key will contain inevitable transmission errors. These must be corrected using an information-exchange not encoded quantum mechanically. This is known as the classical communication.

The gain in security is substantial. Whenever Alice wishes to send Bob a secure message they use QKD to find a new long key, for example 500 bits. This key is so long that exhaustive search will be finished long after the information is out of date. If computing power of the adversary increase, Alice and Bob only generate a longer key. Moreover, the key is only used once, not giving Eve a chance to learn information about the key by matching different encrypted texts. If the information is very sensitive, Alice and Bob can generate enough key to use the *one-time-pad*, described in the next chapter. This encryption technique has been proven unconditionally secure!

This thesis focuses on the codes used for the correction of inevitable errors in the quantum transmission. The codes are implemented into a *protocol* which is a term for rules determining the format and transmission of the data. Compared with the amount of research put into security and experimental setups, little has been done on error correction and key reconciliation. This is mainly due to the fact that research on QKD has been driven by physicists and mathematicians who are experts in quantum mechanics and not in computer science. Papers describing different protocols are often focused on one detail, leaving other questions open. Results of large scale simulations is sparse, to say the least. The aim of this thesis is to study the performance of different protocols for several different important factors, e.g. error-correction efficiency, probability of correction and communication required in the process. The intention is to bring all pieces together and present empirical results of simulations for each factor. The thesis analyzes four different protocols which utilizes different techniques to correct errors.

The layout of the thesis is divided into three parts, Background, Key Reconciliation and Performance Comparison. The Background chapter gives the reader the basic theory required for the understanding of QKD in general and the Key Reconciliation chapter in particular. The last section in Background gives the scope of this thesis. The main chapter is Key Reconciliation, in which an extensive analysis of four different error-correction protocols is given. The following chapter gives a numerical example for comparison of the process of going from raw quantum transfer output to final secure key using the different protocols. The last chapter, Conclusions, can be read independently of the thesis as a review, albeit understanding all parts might be hard without previous knowledge of the field. In the appendix there is a list of notations. It may come in handy when reading the thesis.

The subject of quantum mechanics is often considered very mysterious and far from real life. The cartoon below gives a humoristic approach trying to explain the uncertainty in unknown quantum states, printed with permission from Stephen Notley.


ANGRY FLOWER
angryflower.com
SCHRÖDINGER'S FRIDGE
HEY, IS THERE ANY BEER IN THE FRIDGE?
INDETERMINATE!
I MAY HAVE DRUNK THEM ALREADY - BUT MAYBE I DIDN'T! THUS THE FRIDGE CONTAINS THE POSSIBILITIES OF NO BEERS AND SOME BEERS EXISTING IN SIMULTANEOUS SUPER-POSITION!
THE ONLY WAY TO MAKE THEM REAL IS TO OPEN THE FRIDGE!
OPEN IT!
YEAH!
WHOOSH!
HOLY BLAP!
THE WAVEFORM ISN'T COLLAPSING!
WOM WOM WOM
I'M OBSERVING MY FRICKIN' HEAD OFF, HERE, BUT THEY'RE JUST NOT ACTUALIZING!
I CAN'T DRINK POSSIBLE BEERS!
I NEED ACTUAL BEERS!!!
DAMN YOU, QUANTUM PHYSICS!!!
NEXT WEEK: THE DOUBLE-SLIT EXPERIMENT...
SN+DARREN 02
5-19
The deep and profound mysteries of quantum mechanics.

Chapter 2

Background

2.1 Introduction

Quantum key distribution is a way of transferring a secret key from Alice to Bob using single photons. The photons are send on the *quantum channel.* Eve, who tries to eavesdrop on the quantum channel, will inevitably cause errors in the transmission. Errors also come from imperfections in the setup. After the quantum transfer, Alice and Bob have one string each. This is the raw information from which the final secure key is extracted. The errors need to be corrected for the key to be useful. The correction of the errors is done by error-correcting codes which exchange additional information over a second channel. This is known as the *classical channel.* This channel is an ordinary public communication channel. Public means that the information is not encrypted on the classical channel, i.e. it is available to Eve as well. Eve can gain knowledge of the key from eavesdropping on the quantum and the classical channel. To erase Eve's information, Alice and Bob compress their strings using a hash function. This is called privacy amplification. The compressed output string is the final secure key. The key will be shorter than the original raw output from the quantum channel, but it will be error free and unknown to Eve.

The quantum channel can be based either on fibre or in free space. The public classical channel can be any way of transferring information. The intended setup for this thesis is free-space optical communication for both channels. The objective is to evaluate different error-correction protocols with respect to parameters important for the free-space setup.

In this chapter I will give an introduction to the basic theory and the scope of the thesis. The first section deals with basic mathematics including statistics, information theory and some basic cryptography. The following sections introduce the different steps in Quantum Key Distribution, QKD. The last section gives the scope of this thesis.

I highly recommend the review article by Gisin *et al* which extensively covers the QKD subject [5].

2.2 Basic Mathematical Theory

2.2.1 Statistics

Stochastic variables are commonly denoted with upper case letters and observations from them are commonly denoted with the same letters but in the lower case. Mathematicians favor the letter X, Y and Z for stochastic variables, I will also make use of them. Furthermore, the notation $X \sim D$ is shorthand for "X follows the distribution D".

The probability of an error-correcting code successfully correcting all errors is of great interest for this thesis since a key is useless if it contains errors. This probability will be estimated by simulations. For an estimated parameter to be useful it requires a confidence interval, hopefully a narrow one. For most of my simulations with fixed parameter settings the algorithm corrects all errors in every trail. This makes the standard deviation zero. Unfortunately the standard way of constructing confidence intervals use the standard deviation with multiplicative factors, thus the confidence will be interval zero. Instead I make use of the binomial distribution.

First I define

$$X = \begin{cases} 1 & \text{If the code corrected all erros in a given trial} \\ 0 & \text{Otherwise} \end{cases},$$

with $P(X = 1) = p$. Then the sum of n trials,

$$Y = \sum_{k=1}^{n} X_k,$$

will be binomially distributed with parameters n and p, where p is unknown. The estimate for p is $\hat{p} = \bar{X}$ with X as above. Throughout this thesis, a bar on top of a letter will denote the average value. Based on one observation, y, the lower and upper limits, (p_L, p_U), for a $100(1-\alpha)\%$ confidence interval solves

$$\sum_{k=0}^{y} \binom{n}{k} p_U^k (1-p_U)^{n-k} = \frac{\alpha}{2}$$

and

$$\sum_{k=0}^{y-1} \binom{n}{k} p_L^k (1-p_L)^{n-k} = 1 - \frac{\alpha}{2}.$$

This is made easily with the command `[phat,pci]=binofit(y,n,0.01)` in MATLAB producing $\hat{p}$ and a 99% confidence interval for $\hat{p}$. This means that 99 out of 100 such produced intervals will cover the true p, or in other words; $P(p \in [p_L, p_U]) = 0.99$.

To produce a narrow confidence interval we need to increase n. Figure 2.2.1 below shows the estimated $\hat{p}$ together with a 99% confidence interval for observations $y = 0.5 \times n$ with increasing n. Based of Figure 2.2.1 and my available computing powers, I will use $n > 10^3$.

To create confidence intervals for other observed parameters, I will use the *bootstrap* method. In bootstrap you sort the observations in increasing order. Then you symmetrically choose one lower and one upper observation as limits so

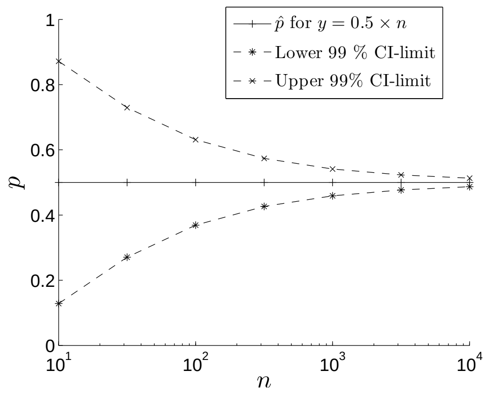


Figure 2.2.1: Estimated $\hat{p}$ with 99% confidence interval.

that $100(1-\alpha)\%$ of the sorted observations lie between the chosen limits. This gives an approximate confidence interval. I will use 99% confidence intervals as standard in this thesis.

2.2.2 Information Theory

Information theory is concerned with the subject of sending information over communication channels. The fundamental problem of this is to reproduce at one point what was sent through a communications channel from another point. Information theory was presented 1948 by Shannon with the first mathematical definition of information [6]. Consider a stochastic variable X for some distribution $p(x)$. Before we observe the value of X, there is a certain amount of uncertainty about its value. After the observation, we have gained information by reducing the uncertainty. We can therefore say that information and uncertainty are related to each other. If we choose X to be the information, it is represented by characters in some alphabet $\mathcal{X}$. Let $\{p(x)\}_{x\in\mathcal{X}}$ be a probability distribution of which character in $\mathcal{X}$ is measured at the receiving point. The distribution $p(x)$ depends on the quality of the communication channel. The entropy of X, denoted $H(X)$, is defined as

$$H(X) = -\sum_{x\in\mathcal{X}} p(x)\log p(x). \tag{2.2.1}$$

Note that all logarithms in this thesis use the base two, unless specified otherwise. $H(X)$ is the average amount of uncertainty in X. If X is uniformly distributed over $\mathcal{X}$, then $H(X) = \log|\mathcal{X}|$. If $\mathcal{X}$ is a set, $|\mathcal{X}|$ denotes the cardinality, i.e. the number of elements in $\mathcal{X}$. In our case $\mathcal{X} = \{0,1\}$ which makes X a Bernoulli trial with $P(X=1) = p$ and $P(X=0) = (p-1)$. This gives the binary entropy function denoted

$$h(p) = -p\cdot\log p - (1-p)\cdot\log(1-p). \tag{2.2.2}$$

Figure 2.2.2 shows the behavior of $h(p)$. The entropy reaches its maximum value of one when $p = 0.5$ and decreases symmetrical to reach zero at $p = 0$ and $p = 1$.

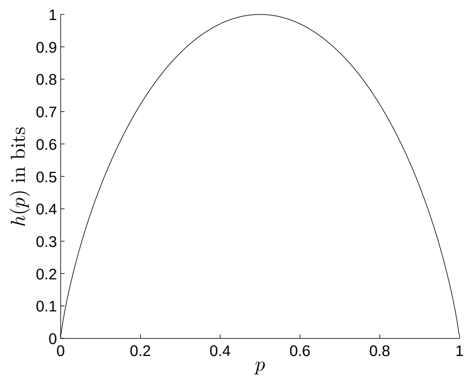


Figure 2.2.2: The binary entropy function

The outcome of X has its maximum amount of uncertainty for $p = 0.5$. Knowing that one outcome has greater probability, there is less uncertainty. For p equal to zero or one, the outcome is deterministic, hence the entropy is zero.

Let A be a string of the outcome from m independent Bernoulli trials with $p = 0.5$, then $H(A) = m$ since $H(A_i) = 1$ and all bits are independent. A *binary symmetric channel*, BSC, models a transmission of a string A independently exposing each bit to noise with the probability p. Each bit takes a Bernoulli trial to determine if it switches value, see Figure 2.2.3 below.

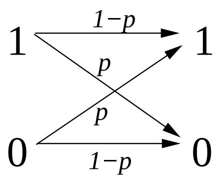


Figure 2.2.3: Schematic binary symmetric channel.

The Shannon capacity of a BSC is defined as

$$C(p) = 1 - h(p).$$

There is a lower limit for the amount of information that needs to be exchanged, i.e. revealed, in order to correct a certain amount of errors in a string sent over a BSC [7]

$$n_{min} = n_{sif} \times [1 - C(p)] = n_{sif} \times h(p), \tag{2.2.3}$$

where n_{sif} is the length of the string after sifting as described later. The probability p in Equation (2.2.3) corresponds to $\bar{e}$, the average bit error rate of the transmission. If Eve disturbs the transmission, there is a risk that the errors will come in bursts. Errors can come from non-uniform noise in detectors as well. This makes the errors non-uniformly distributed. To use the BSC model, Alice and Bob need to agree on a random permutation of their strings before starting the error correction. After this, any binary input/output channel can be regarded as a BSC.

Another important measure in information theory is the mutual information

$$I(X;Y) = \sum_x \sum_y p(x,y) \log \frac{p(x,y)}{p(x)p(y)} = H(X) - H(X|Y). \tag{2.2.4}$$

This measures the amount of information we have about X after having measured Y. $H(X|Y)$ is conditional entropy giving the amount of uncertainty in X knowing Y. If $H(X|Y) = 0$, we have complete knowledge of X giving $I(X;Y) = H(X) = 1$. In our case, let A be Alice's secret key which she transmits to Bob, who measures the string B. If the transmission did not in any way corrupt the information sent, Bob will have $I(A;B) = m$ bits of information. This is equivalent to $A = B$. Between the, two Eve can eavesdrop and learn the string E. Our objective is to keep $I(A;B) > I(A;E)$ and $I(A;B) > I(B;E)$ so that Alice and Bob have more mutual information than Eve does with any of the two.

2.2.3 Cryptography

In cryptography, the sender is commonly called Alice and the receiver is called Bob. The enemy eavesdropping on Alice and Bob's communication is commonly called Eve. Alice has a plain message which she encrypts into a cipher text which she sends to Bob. Eve can eavesdrop and learn the cipher text. Bob has access to the key and can decode the cipher text into the plain message.

The only cryptographic method which is proved to be totally secure is the *one-time-pad* or OTP [5]. The OTP is a symmetrical cipher, i.e. both sender and receiver share the same key. Let m denote the plain message, c the cipher text and k the key. Moreover m_i, c_i and k_i are the characters at place i in messages m, c and k. The message m must be represented in bits. Using the ASCII table each Latin character is one byte represented by eight bits. For example A is represented by the decimal number 65, i.e. the bit combination {01000001}. In OTP k need to be as long as the plain message representation in bits. To send two Latin characters using OTP the key k needs to be 16 bits. The key needs to be totally random and can only be used once or else Eve can get information from combining two cipher texts, hence the name one-time-pad. Given all this the OTP is

$$c_i = m_i \oplus k_i, \tag{2.2.5}$$

where $\oplus$ is addition modulo two. To decrypt the cipher text Bob calculates

$$c_i \oplus k_i = m_i \oplus k_i \oplus k_i = m_i.$$

One definition of the security of the OTP is

$$I(m;c) = 0,$$

using Equation (2.2.4) in the section on information theory. Given the cipher text, we know nothing about the plain message.

2.3 Quantum Key Distribution

2.3.1 Quantum Transfer

The basic idea in QKD is that Alice generates a sequence of truly random bits, each with a value of either zero or one, code the information into single photons and transmit these photons to Bob. Bits encoded in a quantum mechanic way is referred to as a quantum bit, or qubit. This thesis will use the BB84 protocol introduced by Bennett and Brassard in 1984 [3]. In BB84 there are four states for coding the photon denoted

$$\begin{array}{cccc} |\uparrow\rangle & |\rightarrow\rangle & |\nearrow\rangle & |\nwarrow\rangle \\ 0 & 1 & 0 & 1 \end{array}$$

which represent vertical 90°, horizontal 0°, 45° and 135° linear polarization. The bit value 1 is assigned to horizontal and 135° and zero to the other two. The states can be represented as four vectors in two orthonormal bases in two dimensions, see Figure 2.3.1. The bases are denoted H/V or $|+\rangle$ for horizontal/vertical and L/R or $|\times\rangle$ for left/right.

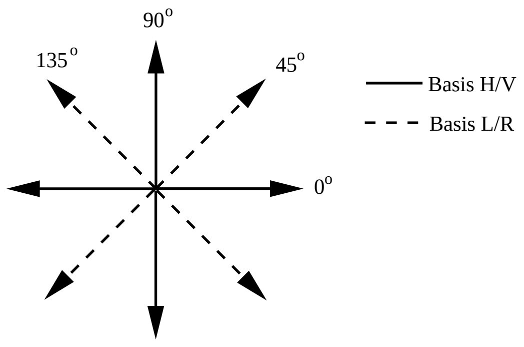


Figure 2.3.1: The two bases in the BB84 protocol

There are three terms to keep separated:

- Bit values - $\{0, 1\}$
- Bases - $H/V = \{90°, 0°\}$ or $L/R = \{45°, 135°\}$
- States - $\{90°, 0°, 45°, 135°\}$

A measurement can be regarded as a quantum mechanic operation projecting the state of the photon onto the chosen basis and giving the eigenvalue of the state in that basis. Due to the overlap of the states the probability of measuring the correct state given the wrong basis is one half. The calculation of the probability is denoted $|\langle State|Basis\rangle|^2$. There are eight cases of projection with two different probabilities of correct outcome:

$$\begin{array}{cccc} |\langle\uparrow|+\rangle|^2 = 1 & |\langle\rightarrow|+\rangle|^2 = 1 & |\langle\nearrow|+\rangle|^2 = \frac{1}{2} & |\langle\nwarrow|+\rangle|^2 = \frac{1}{2} \\ |\langle\uparrow|\times\rangle|^2 = \frac{1}{2} & |\langle\rightarrow|\times\rangle|^2 = \frac{1}{2} & |\langle\nearrow|\times\rangle|^2 = 1 & |\langle\nwarrow|\times\rangle|^2 = 1 \end{array}$$

This can be seen from Figure 2.3.1 as simple projection of a vector in one basis onto a vector in the other. This is the main result giving the possibility of creating a secure key.

The transfer starts with Alice randomly choosing a sequence of bits and basis, encodes this information into photon states and sends the photons to Bob through the quantum channel. Bob, the intended receiver, has no information on which basis to use. He randomly chooses one of the two bases. He will on average select the right basis for half of the photons and measure their states correct. The other half of the photons states will be projected on to the wrong basis, thus giving the wrong measured state for a quarter of the total number of photons. After the quantum transmission, Bob will have a string of length n_{raw}. The number of bits measured correctly by Bob is $n_{raw} \times (\frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2}) = n_{raw} \times \frac{3}{4}$. Therefore Bob's string measured from the quantum channel contains about 25% errors due to measuring in the wrong basis.

The next step is called *sifting*. In the sifting phase Bob publicly announces which basis he used for each photon. Alice then tells Bob if he measured in the right or wrong basis. If Bob used different basis, both parties discard that bit. They never reveal which state the photon was in, i.e. the bit value, but given that Bob chose the right basis to measure in, he will have the same string as Alice. Their strings will on average be of length $n_{sif} = 0.5 \times n_{raw}$ after sifting. If there were no disturbance on the quantum channel, Bob's string will not contain any errors after sifting.

Eve, the adversary trying to make measurements on the quantum channel, can not know which basis to use. Remember that Alice sends truly random states of the single photons, i.e. truly random bits and bases. Eve can only follow the same scheme as Bob and randomly select basis. Thus Eve will only be able to measure three quarters of the string correct. Due to the principles of quantum mechanics it is impossible to make a measurement on an unknown quantum state without disturbing it. Disturbing the state of the photon may be seen as rotating the state vector from Figure 2.3.1 by some angle. If a horizontal state vector is rotated by some small angle, there is a chance of measuring it as a vertical state when using the H/V-basis. Eve can try to send Bob a photon in the same state as she measured the single photon leaving Alice. This is called *intercept/resend* attack and is described further in Section 2.3.4. Another strategy would be to clone the single photon and measure the clones in different bases. However this has been shown to be impossible in quantum mechanics without introducing errors due to inevitable background noise, see for example Gisin and Massar [8]. This is also discussed in Section 2.3.4. Because of the uncertainty in Eve's measurement she will introduce errors that will remain after sifting. If the quantum channel is noiseless, Alice and Bob will know that eavesdropping took place if Bob find errors in his string after sifting. Things are however not so easy in the real world. Limitations in the technology today give rise to a number of other problems. Below I will discuss of some of the more important problems.

There are no efficient true single-photon sources available today. Therefore most experimental setups use faint laser-pulses. Alice decides a state and sends the corresponding laser pulse through a dense optical filter. The number of photons, n, exiting the filter is Poisson distributed

$$P(n) = \exp\left[-\mu\right] \times \frac{\mu^n}{n!},$$

with an average of μ photons per pulse. The chosen μ needs to be smaller than one, since there is a non-negligible probability of sending more than one photon. For example $P(n \geq 2|\mu = 0.1) \approx 0.05$. A pulse with more than one photon opens a window for Eve to gain complete knowledge of this bit's value. She can for example split the multiple-photon pulse in two using a beam splitter, sending one photon to Bob, and keeping the others unmeasured. This is known as *photon-number-*

splitting or PNS attack. Bob will not discover this attack since he receives an undisturbed photon as normal. Let us assume that Eve has unlimited resources, allowing her to keep an arbitrary number of photons undisturbed for arbitrary long time. Then Eve can measure the stored photon using the right basis after Alice and Bob publicly announced it, thus getting full information of the bit value. Therefore Alice and Bob need to minimize the number of multiple-photon pulses by choosing $\mu < 1$. Commonly $\mu = 0.1$ is used. However $P(n = 0|\mu = 0.1) \approx 0.9$, resulting in that most of Alice's pulses contain no photons.

To compensate for the decrease in transfer speed due to the low μ, Alice can increase her pulse repetition frequency, PRF. Since it is not deterministic when Bob will receive pulses containing photons, Alice and Bob need a way to distinguish between the pulses so empty ones can be discarded. This is done with *time windows* in the measuring equipment. A time window is open for a short fixed time when the photon arrivals are expected and then closed for a longer fixed time before opening again. Alice sends one signal per open time window. Bob only measures during an open time window. In the sifting phase, Bob first announces in which time windows he detected photons, Alice discards all other bits. The shorter the time window, the faster the transmission can be made. The concept of time windows also helps compensating for dark counts in Bob's detector. A dark count is when the detector reports a photon detection without a photon actually impinging on it. To set the length of the time window Alice and Bob needs to consider several factors:

- The time window needs to be open longer than the pulse length, so there is no risk of missing the photon in the pulse due to time window closing too early.
- The closed time between two open time windows needs to be longer than the time the detector takes to reset. This is known as the *dead time* of the detector. If this is not done, a photon might impinge on a non-active detector and thus be missed.
- The length of the time window should not be larger than the period set by Alice PRF, i.e. one open window per incoming pulse.
- To lower the probability of dark counts in Bob's detector inside time window, the window should be made as short as possible.

There are lasers with a pulse length down to femtoseconds and lasers with PRF in the GHz range available. Therefore the limitations today comes from the electronics, both synchronizing and detector's dead time. The shorter the time window, the higher is the demand for accurate time synchronization between Alice and Bob. Since pulse lengths can be very short, it is of great importance that Alice and Bob have accurate synchronization. If synchronization fails, Alice and Bob have no way of knowing which bits to save and which to discard.

A dark count happens when Bob's detector gives a signal without an impinging photon. The probability of dark counts in a time window can however be controlled. I let p_{dc} denote the probability of dark count occurring during an open time window. The number of dark counts are usually given as total number of counts per second (cps). The shorter the time windows are open, the lower is the chance of a dark count happening inside them.

Noise can come from other sources such as:

- Background noise from natural light sources, e.g. the sun.
- Imperfections in optics and electronics.

Background noise is mostly a problem during daytime operations. Reports from free space experiments claim that during the day, a significant amount of the errors comes from background noise [9]. It can however be filtered out with spatial filtering and narrow bandwidth filters at the wavelength of the transmitting laser. Imperfections in the optics of the setup can not be easily improved once a system is built. Noise in electrical components can be very hard to eliminate. These other noise sources are however only minor problems compared to the dark counts. The total bit error rate, BER, in Bob's string will be $p_{dc} + p_{noise} + p_{Eve}$, where p_{Eve} is the errors introduced by Eve during eavesdropping. BER is the number of errors divided by the length of the string. I will denote the bit error rate in Bob's string by e.

The removal of errors is done in the error-correction phase. To reduce the information Eve learned from eavesdropping, the key is shortened in the privacy-amplification phase. This gives a final key of length $r < n_{sif}$. The length depends on which error-correction code is used and desired level of security after the privacy amplification, more on this in Sections 2.3.2 and 2.3.3. Figure 2.3.2 illustrates the steps in the quantum transfer. The schematic shows Alice sending a pulse in each time window. The actual number of photons sent is marked as arrows representing the state of the photons sent trough the quantum channel. Bob's detector gives two dark counts, one outside time windows and one introducing an error. The dark counts are marked as crosses on the time window line and produces two detector clicks marked as arrows. After the quantum transmission, sifting, error correction and privacy amplification is performed. These steps use the classical channel for communication.

To illustrate with an example which uses some rough estimations. The number of unsifted bits exchanged per second is roughly estimated by

$$n_{raw} = \eta \times P(n \geq 1) \times T_{atm} \times T_{opt} \times PRF_{Alice},$$

where η is the detector efficiency, T_{atm} the transmission through atmosphere and T_{opt} the transmission in the optics. The atmosphere has a transmission window at 850 nm. At this wavelength T_{atm} is about 0.8 for a path of one kilometer and the estimation of T_{opt} is 0.1. The company *PerkinElemer* manufactures a single-photon counter module with 250 dark counts per second, a dead time of 50 ns and $\eta \approx 0.45$ for the 850 nm wavelength. Assume we obtain a time-window opening of 10 ns followed by 50 ns closure with the synchronizing electronics, thus making one period 60 ns long. This gives $1/60 \cdot 10^{-9} \approx 1.67 \cdot 10^7$ time windows possible per second. Furthermore, we assume that Alice has a laser fast enough to utilize full capacity and uses $\mu = 0.1$. Then Bob will receive $0.45 \times 0.1 \times 0.8 \times 0.1 \times 1.67 \cdot 10^7 \approx 54 \cdot 10^3$ non-empty pulses per second, giving equally many unsifted bits. This is made with the assumption that Bob can not tell a one-photon click from a multiple-photon click. After sifting Alice and Bob will have $n_{sif} = 27 \cdot 10^3$ bits per second. The p_{dc} would be

$$\frac{250}{\frac{1\,s}{(10+50)\,ns}} \times \frac{10\,ns}{(10+50)\,ns} = 2.5 \cdot 10^{-6}.$$

There will thus be about 40 dark counts in Bob's string.

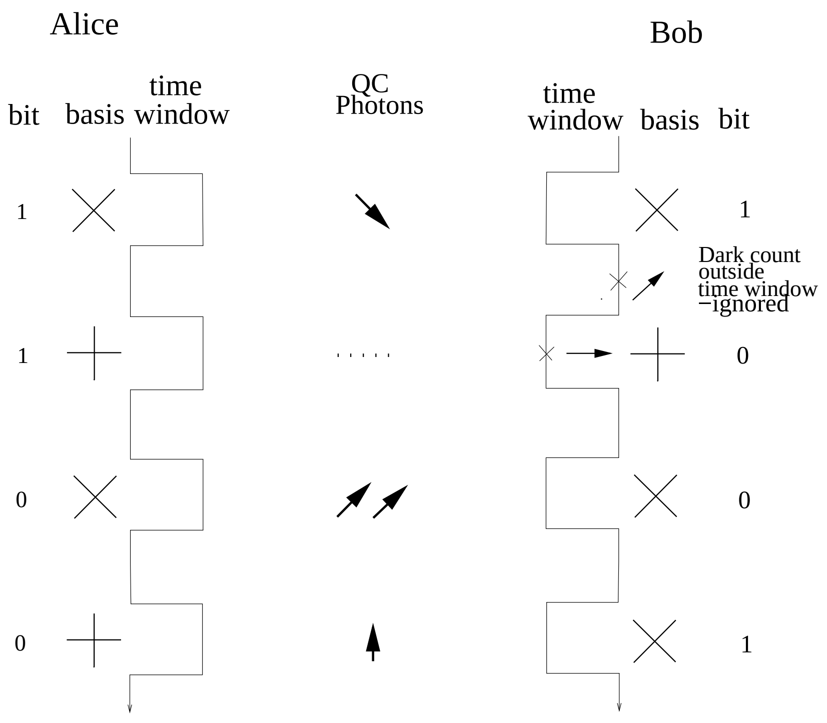


Figure 2.3.2: The essential steps in the quantum transfer.

The paper by Hughes *et al.* contains detailed discussion of similar estimates as above for a real 10 km free space QKD experiment together with results from the experiment [9].

2.3.2 Key Reconciliation - Error Correction

After the sifting, there will be errors in Bob's string. These errors come from Eve's eavesdropping and transmission errors, i.e. as described above. The errors must be corrected if the final key shall work properly. The name key reconciliation means that Alice and Bob create identical keys by detecting the errors in Bob's measured string. This is done by error-correcting codes. I use the term error correction albeit the errors can be discarded instead of corrected. Discarding the errors decreases key generation ratio, but reduces the information leaked to Eve during the process.

Classical optical transmissions-links have an error rate down to 10^{-9}, hence only a small portion of packages sent will contain errors. These can be detected by check sums as used in most digital communication. If a package is found to be erroneous, it is resent. In QKD the error rate is typically reported to be a few percent. To achieve the same error rate as in classical optical communication we would need to send a string with five percent error rate seven times, i.e. $0.05^7 \approx 10^{-9}$. This is however not an option since the security relies on truly random bits. Resending the sequence eliminates the possibility to extract a secure key.

Bob counts all errors he corrects so that the actual error rate of the starting string is known after the correction phase. If it is considered to be too large for the key to be secure, he and Alice discard the key. Therefore there is no risk of accepting an insecure key just because the starting estimation of the error rate was to low.

The error-correction code exchanges information over the public channel, thus exposing it to Eve. I will assume that the public channel is authentic, i.e. Eve can listen but not tamper with the sent information. Since Eve can listen to the correction information, she will gain partial information about the key. The amount is modeled by a function $f(x) : \{0,1\}^{n_{sif}} \to \{0,1\}^l$, thus giving Eve l bits of information. These bits can be either physical information of the value of one bit or the parity of a block of bits. The function f depends on which error-correction code is used. The less information we exchange, the better. Remember that Equation (2.2.3) gives the lower limit. This limit is called the Shannon limit.

The leakage of information creates the need of enhancing the security, see the Section 2.3.3 on privacy amplification below.

2.3.3 Privacy Amplification

After the quantum transmission and the key reconciliation, a proportion of the secure key might have been leaked to Eve due to her eavesdropping. This amount depends on her strategy of eavesdropping the quantum channel and which error-correction code used. The number of bits leaked in the error correcting depends on the number of errors to correct, i.e. the error rate e. To achieve security, Alice and Bob will assume that all errors are introduced by Eve, thus leaking her information. Eve's information on the final key will be upper bounded by the level of security chosen. The method of reducing Eve's information of the final secure key to an arbitrary small amount is called privacy amplification, as introduced by Bennett *et al.* [10]. They used the concept of *universal hashing*, following the ideas and notation from Carter and Wegman [11].

Definition 2.3.1. The class $H = \{g : \{0,1\}^i \to \{0,1\}^j, i > j\}$ is $universal_2$ if for any two distinct strings $x, y \in \{0,1\}^i$ the number of $g \in H : \{g(x) = g(y), x \neq y\}$ is less than or equal to $1/2^j \times |H|$.

The '2' in $universal_2$ comes from x being a binary string. If we then choose g from H according to the uniform distribution, the probability that $g(x) = g(y)$ given that $x \neq y$ is upper bounded by $1/2^j$ for $g : \{0,1\}^i \to \{0,1\}^j$. I choose the function called H_3 [11]. The property of Definition 2.3.1 was empirically validated for my implementation of H_3. The hash function spreads the input in a chaotic fashion, as little as one bit-error in the in-string will multiply and produce a significantly different out-string. This is the idea of privacy amplification.

When using privacy amplification, we shorten our partially secure string, x, by an amount depending on our estimation of Eve's knowledge about the string and a security factor, s. This makes Eves information exponentially decreasing in s. Assume Eve gains at most k bits of information from eavesdropping on the quantum channel according to some function $e(x) : \{0,1\}^{n_{sif}} \to \{0,1\}^k$. The function $e(x)$ depends on her choice of strategy. She also learns at most l bits according to $f(x) : \{0,1\}^{n_{sif}} \to \{0,1\}^l$ from the key reconciliation phase depending on which error-correction code was used. Let $r = n_{sif} - k - l - s$

be the length of the final key and choose $g \in H_{universal_2} : \{0,1\}^{n_{sif}} \to \{0,1\}^r$ randomly from H. Since g is transmitted via the public channel, it will also be known to Eve. Denote the final secure key by $K = g(x)$, which is of length r. Eve's knowledge about $e, e(x), f, f(x)$ and g is summed up in V. Then according to Bennett and Brassard [10] Eve's information is upper bounded by

$$I(K;V) \leq \frac{2^{-s}}{\ln 2}. \tag{2.3.1}$$

If we choose $s = 20$, Eve will have as little as less then $1.4 \cdot 10^{-6}$ bits of information about K. This is of course given that we have chosen k and l properly. Lütkenhaus gave the estimate $(l + k) = \tau(\bar{\epsilon}) \times n_{sif}$ where $\tau(\bar{\epsilon}) = \log(1 + 4\bar{\epsilon} - 4\bar{\epsilon}^2)$ [7]. When correcting errors and leaking error positions in the error-correction phase, we have $\bar{\epsilon} \approx e$. The estimate uses the restriction that Eve does not use *coherent* or *collective* attacks and that Bob's detector measures all states with equal efficiency.

Figure 2.3.3 displays a sketch over the amount of information that Alice, Bob and Eve have in the different steps in QKD using the notation from previous sections.

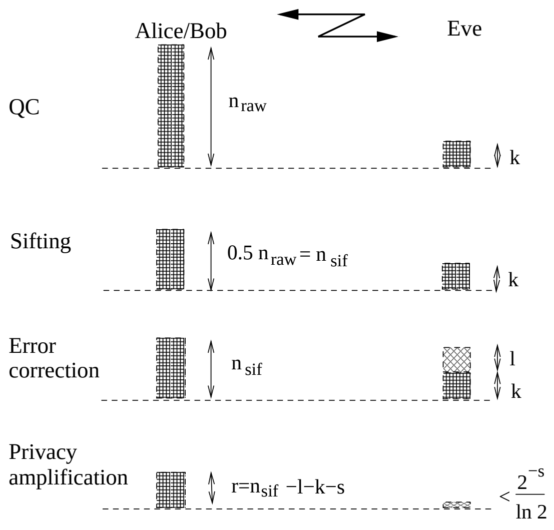


Figure 2.3.3: Schematics over the amount of information available to the different parties.

2.3.4 Security Aspects

Introduction

Eve can launch several different attacks on a QKD system. Figure 2.3.4 shows a schematic picture of Eve having control of both the quantum and the classical channel. Think of the box named Eve in Figure 2.3.4 as the collection of resources available to Eve allowing her to perform different attacks. The subsections below deal with some of the different attacks in Eve's arsenal.

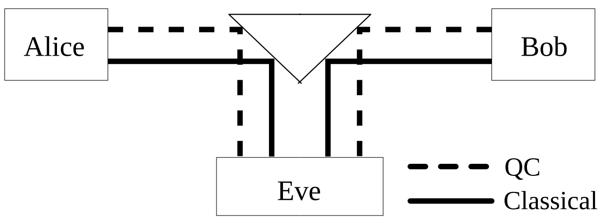


Figure 2.3.4: Eve attacking both channels.

A QKD system is very vulnerable to the *man-in-the-middle* attack. Here Eve has taken control of both the quantum channel and the classical channel. She can perform measurements on the quantum channel which will be described below and forge messages on the classical channel. Alice believes she is talking to Bob and Bob believes he is talking to Alice but they are both talking to Eve mimicking the other party. Eve use QKD to distil one key between her and Alice and another between her and Bob. Alice and Bob uses their keys to send encrypted messages to what they think is each other, but is really Eve who can decrypt their messages. This threat calls for a way to ensure Alice and Bob that they are in fact talking to each other, i.e. they need to authenticate the classical channel. Authentication is described below.

Authentication

One way to authenticate the classical channel is for Alice and Bob to share a short initial secret string. Using this string together with a hash function they can create a tag from each message they wish to exchange. The other party will only accept the message if he computes the same tag using his initial string. After one round trip of QKD, Alice and Bob use some of their new key for authentication in the next round trip. This is why quantum key distribution is sometimes referred to as Quantum Key Growing. The issue of authentication with implementation on QKD is explained in detail by Cederlöf [12].

Eve's Eavesdropping Strategies

Eve's attacks are divided into two main classes: *individual* and *coherent* [5]. In a coherent attack Eve possesses the ability to measure several photons at the same time. She can also use multiple probes on one photon. The beam splitting example in Section 2.3.1 above, where Eve stores a number of photons and after sifting measures all of them, is an example of a coherent attack. The example is actually a special case of coherent attacks known as *collective* attacks. Eve only uses one probe per photon, but measures all of them coherently.

Individual attacks are performed immediately on a single photon in the quantum channel. I will describe three individual attacks.

Intercept/Resend

In *intercept/resend* attack, Eve performs measurements on every photon leaving Alice. She then sends a photon to Bob in the same state as her measured outcome. Eve can measure the photons in an arbitrary basis, most commonly discussed are the *canonical* and *Breidbart* basis [5, 4].

The two canonical bases are the normal rectilinear basis used by Alice and Bob. The probability of Eve measuring correct bit value is 0.75. Measuring in this basis will give Eve a mutual information of $I(A_i; E_i) = 1 - 0.5 \times h(0.5) = 0.5$ per bit i [5]. Remember that $h(p_{A_i}) = 1$ since A_i is chosen uniformly from $\{0, 1\}$ making $p_{A_i} = 0.5$. Her total mutual information will be $I(A; E) = 0.5 \times n_{sif}$ since Eve also will know which bits were discarded in the sifting. Eve will on average introduce 25% errors in Bobs string, i.e. the bits she measured wrong.

The Breidbart basis is an intermediate basis between the two canonical bases. In this case, the probability of Eve measuring the correct value is about 0.854. The drawback is that her mutual information will be $I(A_i; E_i) = 1 - h(0.854) = 0.399$. In the canonical basis, Eve's information will be deterministic in half the cases but in the Breidbart basis she will measure correct with $p \approx 0.85$ in each case, or mathematically speaking using Equation (2.2.4)

$$h(0.5) - 0.5 \times h(0.5) > h(0.5) - h(0.854).$$

The intercept/resend strategy gives Eve $k = 0.5 \times n_{sif}$ for canonical basis and $k = 0.399 \times n_{sif}$ for the Breidbart basis.

Gisin *et al.* stated in [5], that for a optimal symmetric individual intercept/resend attack Eve's information is:

$$I^{max}(A; E) = 1 - h(p) = 1 - h\left(\frac{1 + \sin\theta}{2}\right),$$

where p is the probability of Eve measuring the correct outcome. The parameter θ defines which basis she measures in. In this strategy, the fidelity of photons sent from Alice to Bob will be $\mathcal{F} = 1 - \mathcal{D}$ with the original states after passing through Eve's attack. $\mathcal{D}$ is equal to

$$\mathcal{D} = \frac{1 - \cos\theta}{2}$$

This makes $\mathcal{D}$ equal to Bob's error rate. The mutual information between Alice and Bob is

$$I(A_i; B_i) = 1 - h(e) = 1 - h(\mathcal{D}).$$

By choosing θ, Eve can control the amount of errors she introduces. Using $\theta = 0$ will give $\mathcal{D} = 0$ and thus $I(A; E) = 0$. The Breitbart basis is equal to $\theta = \pi/4$ giving $\mathcal{D} = 0.1464$ and

$$I(A_i; E_i) = 1 - h\left(\frac{1 + \sin\pi/4}{2}\right) = 0.399,$$

as before. The drawback is that we also have that

$$I(A_i; B_i) = 1 - h(0.1464) = 0.399.$$

Thus Alice and Bob can not distil a final secure key if $e \geq 0.146$, since then Eve may have more mutual information. The tolerated error rate will be lower still, if we assume that Eve can perform even better attacks.

Before the error correction starts, Alice and Bob can publicly announce a number of random bits to estimate e. These bits are discarded. If Alice and Bob find e equal to 0.25, they know eavesdropping occurred on the quantum channel.

Remember that Alice and Bob assume that Eve caused all errors. Alice and Bob can abort and try again, hoping that Eve will not interfere a second time. A better choice for Eve would be to only attack a fraction of the photons. This will reduce the error rate induced, but it will also reduce her gain of knowledge by the same fraction. This way Eve will increase her chance of remaining undiscovered.

Quantum Cloning Attack

If we assume that Eve has unlimited resources, she might possess the technique to clone photons. It has been shown that it is impossible to perfectly clone a photon, see for example Gisin and Massar [8]. Think of a cloning device as cloning N input photons into M output photons by having $M-N$ blank copies. All input photons are in the same state and all output photons are in the same state. In a perfect cloning machine all input and output photons would be in the same state. Gisin and Massar use the fidelity, $\mathcal{F}$, to measure the quality of the cloning. The fidelity is the average overlap between any of the identical output copies and the input state. If the copy is in the exact same state the overlap will be one and if the copy is in a orthogonal state the overlap will be zero. The optimal fidelity achievable still obeying the principles of quantum mechanics is

$$\mathcal{F}_{N,M} = \frac{M(N+1)+N}{M(N+2)}.$$

The fidelity of cloning one photon into two is thus $\mathcal{F}_{1,2} = \frac{5}{6}$. For example, given the state $0°$ on the input photon there is a non-negligible probability of measuring the output photon in a different state even if measured in the H/V basis. This prevents Eve from perfectly cloning the photons send by Alice, Eve will always introduce errors. If Eve uses an optimal cloning device, the states of the photons exiting from Eve will have an overlap of 5/6 with the state of the photon sent by Alice. The chance of Bob measuring the wrong state even if using the correct basis due to this non-perfect overlap is $1-\mathcal{F}_{1,2} = 1-5/6 \approx 0.167$. Thus, an error rate of 0.167 is enough to suspect the use of quantum cloning attack.

Photon-Number Splitting

By inserting a partially transparent mirror in the path between Alice and Bob, Eve can split multiple-photon pulses into two beams. One part of the pulse goes to Bob and the other is hers to make measurements on. Quantum mechanic principles allow Eve to count the number of photons in pulses without disturbing their states. She has the choice to only insert the mirror if there are more than one photon, so she does not risk to disturb single-photons pulses. In links with high transmission losses another strategy would be to block all single-photon pulses and send one of the photons of multiple-photon pulses to Bob and measure on the remaining. She must then also insert a lossless link to Bob, so he does not detect the extra attenuation of the channel. Bob will only receive pulses for which Eve also will learn the states. This gives her full knowledge of the final key, assuming she can store the photons until the bases are revealed in the sifting.

The assumption that Eve learns all bits with multiple-photon signals gives her the fraction $[1-P(n \leq 1)] = 1 - (\exp[-\mu] + \mu \times \exp[-\mu])$ of n_{sif}.

The technology to store a photon undisturbed for arbitrary long time is distant. Today the coherence time is considerably less than one minute. To prevent a PNS attack with today's technology Alice and Bob simply waits longer than Eve's coherence time before the sifting. Thus Eve's captured photons have changed states and are useless. However, longer coherence time is required in quantum computers and is therefore a hot research field.

Analysis of Security

The proof of unconditional security is hard and tedious. Gisin [5] found the bound $e \approx 0.15$ for individual attacks and $e \approx 0.11$ for coherent attacks. Assuming that Eve induces all errors, larger e than these bounds would give her too much information for the key exchange to be considered secure. The bounds were derived using assumption of one-photon source being available to Alice. Inamori *et al.* [13] states a security proof dealing with among other things, weak laser-pulses. Following the limitations for the QKD steps stated in the proof, the conditional entropy of Eve's total information of the final key K given her complete knowledge V is

$$H(K|V) \geq |K| - \varepsilon(n_{sif}, |K|)$$

where ε is exponentially decreasing in n_{sif} and $|K|$. If $H(K|V) = |K|$ Eve has minimum amount of information and can only hope to guess the key.

The unconditional security of QKD is a very hard question to solve. The proofs often assume technology which today is out of reach for all the three of Alice, Bob and Eve. Practical limits of today's technology work both for and against Alice and Bob. Although efficient single-photon emitters, which will prevent photon-number-splitting, are not available yet, the technology today does not allow Eve to store photons undisturbed for arbitrary long time either.

2.4 The Scope of the Thesis

The scope of this thesis is to analyze the performance of different protocols in the classical communication of a QKD system, i.e. the key reconciliation and the privacy amplification. The purpose of the work is to find suitable protocols that can be used in a tactical free-space QKD link proposed by FOI (Swedish Defence Research Agency) and analyze the performance. Therefore, this thesis is constrained to investigate protocols appropriate for free-space QKD links, where the requirements and limitations are somewhat different compared to fibre based systems. Accessibility and transfer rate of the classical channel is seldom a problem in a fibre system since the classical channel is normally implemented on the same fibre as the quantum channel. These features are not necessary true for the classical channel of free-space QKD links. Therefore, the amount of information transmitted over the classical channel and the interactivity between Alice and Bob are of more importance when analyzing the performance of a free-space link. The results of this thesis are of cause applicable for fibre systems as well. However, special limitations for fibre systems are not taken in to account in the analysis of the performance.

The tactical free-space QKD link proposed by FOI is intended to be used between two mobile units, e.g. between vehicles or vessels, separated by a distance of maximum a few kilometers. Free-space QKD has been demonstrated over a distance of 23 km [14]. Stationary links, such as telephone lines and optical fibres, cannot be used for the classical communication since the units are mobile.

FOI will implement the classical channel on a free-space laser link. Although such a link is more complicated than for example a radio link, there are several important advantages with a laser link. The transfer speed can be substantially higher on an optical link. Furthermore, the beam of a laser is narrow and directed in contrast to radio transmission or even microwave links. Therefore, it is very difficult eavesdrop or even notice a laser link. However, the most important reason FOI has chosen laser link for the classical communication is that the strong laser link can be used for alignment of the quantum channel. The two channels will operate along the same path but at different wavelengths, e.g. 850 nm and 1550 nm, where the atmosphere has transmission windows. It is demanding to keep a high accessibility of an optical link since it is sensitive of disturbance such as vibrations of the platform and turbulence in the atmosphere. With the very weak signal on the quantum channel, this task becomes even more demanding. Therefore, a strong laser link for the classical channel can be used to maintain the alignment for both the channels. Moreover, FOI has chosen the BB84 protocol as described above. Figure 2.4.1 below shows schematics of the proposed free-space QKD link. Like most of today's experimental setups, FOI will also use weak laser pulses. As expected transfer speed the thesis will use $n_{raw} = 100$ kbit/s. This gives the norm of $n_{sif} = 50,000$. This assumption is with today's technology optimistic, but probably achievable in the near future.

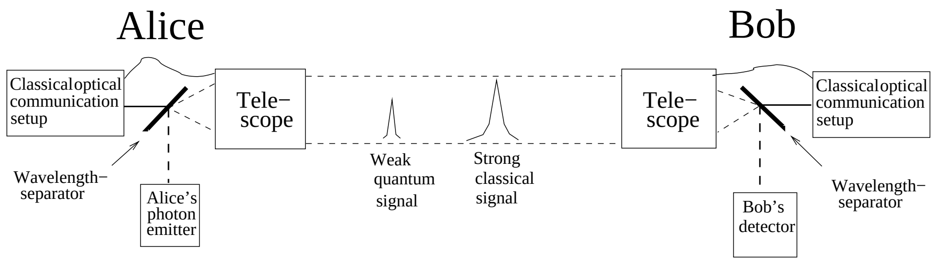


Figure 2.4.1: Schematics over the free-space setup of QKD.

As mentioned above, the limit for tolerable e is about 11%-15%. Experiments typically reports an error rate of about $6 \pm 3\%$. I will use $\bar{e} = \{0.01,\ 0.03,\ 0.05,\ 0.06,\ 0.08,\ 0.1,\ 0.15\}$ to cover the interesting interval.

The important measure is the key generation ratio

$$R = \frac{r}{n_{raw}} \times \beta, \tag{2.4.1}$$

where r is the length of the final key after error correction and privacy amplification and β as below.

I use the notation β for the probability of correcting all errors. This corresponds to p in Section 2.2.1. If an error-correction code only succeeds in half the cases, i.e. $\beta = 0.5$, it will half the key generation ratio. Because of this, β is included in Equation (2.4.1).

Moreover I will comment on the amount of non-leaking information exchanged, such as random permutations and so on. There are many ways to implement a protocol for the classical optical communication. I model the total amount of information as a function of desired information to send, N, with

$$N_{total} \approx \alpha_0 + \alpha_1 \times N.$$

The parameter α_0 is redundant information such as start bits, stop bits and checksum bits. The other parameter, α_1, is the representation of the information desired to send. Both parameters will of course vary with both the size and type of N, but I assume them to be approximately constant. My point being that it is better to send bits as one package instead of one at a time. With larger packages, α_0 will become a lower percentage of N_{total}. I make no intention of estimating α_0 and α_1, I just use the model to help the discussion of the communication in the different protocols.

Chapter 3

Key Reconciliation

3.1 Introduction

I have chosen the name key reconciliation since the main task of this step is to ensure that Alice and Bob share identical keys. In my implementations all errors are corrected, instead of discarded. This preserves the length of the key, but may leak more information to Eve.

The four protocols chosen for this thesis is the Cascade protocol, Yahmamura and Ihizuka's protocol, Low Density Parity-Check Codes and the Winnow protocol. They represent different techniques of performing error correction. Many other protocols are modifications of primarily Cascade, e.g. with introduction of Low Density Parity-Checks. I have chosen to examine the original protocols since understanding of modified protocols requires understanding of the original. Also, we need to know the original performance to see if the modified performs better. The combination and modification of protocols is left as experiments to the interested reader.

To test the different methods, I have chosen a Monte Carlo strategy. Trials were performed by randomly creating Alice's string A and then introducing errors to create Bob's string B. Then the protocol corrected the strings. The errors were introduced by making a Bernoulli trial for each bit with $p = \bar{e}$ being the probability of an error, i.e. the mean error rate. This makes the actual error rate, e, differerent from the current $\bar{e}$. Since the error rate effectively is a sum of Bernoulli trials, only scaled with sample size, it will binomially distributed.The length of the binary strings, n_{sif}, is large enough to use the normal distribution approximation. Hence the actual error rate is normally distributed around $\bar{e}$. For Low Density Parity-Check Code the error rate was fixed, but the errors were introduced in random positions.

Some codes use random functions as a part of the correction. Computers can create true random numbers via extra hardware, but not via software. Software-generated "random" numbers are therefore called *pseudo-random*. These are created from functions using an initial starting number called *seed*. For every seed they give a purely deterministic series of numbers, but spread in a very chaotic way mimicking true random numbers. The beauty of it is that Alice and Bob are seldom dependent of true random numbers for security in error correction. They only need to agree on the seed to create exactly the same "random" series using, assuming of course that Alice and Bob use the same pseudo-random generator function. This drastically decreases the amount of classical information that need to be sent. I will base all random numbers on a (0,1)-float generated with the standard function.

I have implemented Cascade, AYHI and Winnow in Sun's *Java*, version J2SE 5.0, with *Eclipse* as compiler and executer. For LDPCC I used MATLAB because I found prefabricated codes for LDPCC to use in MATLAB.

3.2 Cascade Protocol

3.2.1 Introduction

The Cascade protocol was suggested by Brassard and Salvail [15]. It is designed to work close to the theoretical Shannon limit. The protocol uses interactive communication between Alice and Bob and works by the principle of comparing parities between blocks of key-bits. This enables detection of blocks with odd number of errors. When such a block is found, a binary search inside the block reveals the position of an odd error. The protocol works in a arbitrary number of passes, each pass uses different block partitions. After each pass, each block contains an even number of errors or no errors. If an error is found in one block in pass i, the algorithm tracks the bit back to it's blocks in passes $1, \dots, i-1$. By correction of the bit, there will now be a blocks with odd number of errors in passes $1, \dots, i-1$. Binary searches find these errors, possibly creating more blocks with odd number of errors. This is continued until no blocks have an odd number of errors. Then the protocol proceeds to the next pass and creates new blocks and so on. The key to a good performance in the Cascade protocol is the how to partition the string into blocks in each pass. The size of the blocks depends on the bit error rate, e.

I have implemented the Cascade protocol proposed by Liu [16], called $Cascade^{opt}$. Liu calculated the first two block-partitioning parameters k_1 and k_2 which maximizes the number of errors corrected in the first and second pass, since most errors are detected in these passes.

3.2.2 Algorithm

The algorithm uses a function, in this section denoted f, which randomly maps the bits in Bob's string into a number of smaller blocks. The number of blocks in pass i is $\lceil n_{sif}/k_i \rceil$. In my implementation, f is an array with same length as the original string containing elements representing to which smaller block the particular bit in the original string should be mapped. For mapping $A = \{1,\ 2,\ 3,\ 4,\ 5\}$ into three smaller blocks one array possibility is

$$\{2,\ 1,\ 1,\ 3,\ 2\}$$

which gives

$$f(A) = \big\{\{2,\ 3\},\ \{1,\ 5\},\ \{4\}\big\}.$$

To decide block lengths in each pass, the algorithm uses an estimate of the e. Liu states that k_1 and k_2 should be chosen to minimize [16]

$$\frac{n}{k_1} + \frac{n \times P_{odd}}{k_1} \times \log k_1 + \frac{n}{k_2} + \frac{1}{2} \times \left(n \times \bar{e} - \frac{n \times P_{odd}}{k_1} \right) \times (\log k_1 + \log k_2)\,, \quad (3.2.1)$$

where

$$P_{odd} = \frac{1 - (1 - 2 \times \bar{e})^{k_1}}{2}.$$

Liu also states $k_3 = 2 \times k_2$ and $k_4 = 2 \times k_3$.

I have chosen to use the four passes, the same number as Brassard & Salvail and Liu used. Choosing one more pass will help correction, but reveal more parity information to Eve. We want to keep the number of passes as low as possible and still correct all errors.

The Cascade algorithm is

```
procedure Cascade (A : Alice's bit string
                  B : Bob's bit string ) is
begin
  – Decide $k_1, \dots, k_4$ based on the estimate for $\bar{e}$.
  – Decide on one seed for creation of the random permutation function, $f_i$.
  – Alice does following:
      Divide A into $\lceil \frac{n_{sif}}{k_1} \rceil$ blocks of size $k_1$ and collect them in $Set_1$.
      Create $f_2$ using the seed and collects the $k_2$ blocks from $f_2(A)$ in $Set_2$.
      Repeat procedure to create $Set_3$ and $Set_4$.
  – Using $Set_{1,\dots,4}$, Alice creates a matrix $\mathbf{P}$ with the parities of all her blocks.
      Row 1 will be the parities of the blocks in collected in $Set_1$.
  – Alice sends the parity matrix to Bob.
  – Pass 1:
      Bob creates $Set_1$ by dividing B into $\lceil \frac{n_{sif}}{k_1} \rceil$ blocks of size $k_1$.
      Calculates the parities of the blocks in his $Set_1$
      and compares to the parity matrix $\mathbf{P}$ sent by Alice.
      If a mismatch in parity is found, Bob and Alice run
      a binary search within the block to find the
      position of the odd error. This way Bob can
      correct all odd errors in the blocks in $Set_1$.
  – Pass i=2,. . .,4:
      Bob uses the seed to create $f_i$ which defines
      $Set_i$ of new blocks according to $f_i(B)$.
      Comparing his parities with row $i$ of Alice's parity matrix
      Bob collects all blocks in $Set_{2,\dots,i}$ with parity mismatch
      in a set called $\mathcal{K}$.
      He notifies Alice which blocks are in $\mathcal{K}$,
      and Alice also creates the set $\mathcal{K}$.
      Starting with the shortest block
      in $\mathcal{K}$, Bob and Alice run a binary search to find the
      erroneous bit. Let this bit have index $j$.
      Bob corrects this bit.
      Then they collect all blocks in passes $1, \dots, i$
      that contains the bit $j$ in the set $\mathcal{B}$.
      Then calculate the set $\mathcal{K}' = (\mathcal{B} \bigcup \mathcal{K}) \backslash (\mathcal{B} \bigcap \mathcal{K})$
      which will be the set of blocks with parity mismatch after correcting $j$.
      Set $\mathcal{K} = \mathcal{K}'$ and run again for the shortest block.
      This is repeated until $\mathcal{K} = \emptyset$.
      Then go to pass $i + 1$.
end Cascade
```

3.2.3 Simulation Results

The simulation was done with the following parameters

- Number of sifted bits, $n_{sif} = 10,000$
- Number of trials per each mean error rate, $M = 10,000$
- Original mean error rates : $\bar{e} = \{0.01,\ 0.03,\ 0.05,\ 0.06,\ 0.08,\ 0.10,\ 0.15\}$

The number of sifted bits was chosen to agree with Brassard and Salvail [15]. The number of simulations was chosen as large as possible, but still small enough for the simulation to finish in reasonable time. The introduction of errors in Bob's bit string was implemented as described in the introduction above. The following data is presented in the Table 3.2.1. Both confidence intervals are calculated with bootstrap. N denotes number of revealed parity bits for one trial.

- n_{min}, the average minimum number of revealed information needed to correct an error rate of $\bar{e}$, according to Equation (2.2.3).
- $\bar{N}$, the mean total number of parity bits revealed in the correction. This includes $\bar{N}_{parity}$.
- $\bar{N}_{parity}$, the mean number of parity bits revealed in the binary searches. These bits are the interactive communication, thus can not be sent in advance as a parity matrix.
- $[N_L, N_U]$, an approximate 99% confidence interval for the spread of N around $\bar{N}$.
- $[e_L, e_U]$, an approximate 99% confidence interval for the spread of actual e around $\bar{e}$.
- $[\#]Failure$, Number of trials for each simulation where Cascade was unsuccessful to correct all errors.

I calculated k_1 and k_2 with numerical optimization for $k_1, k_2 \in [1, 1000]$. The interval was set after graphic inspection of the behavior of Equation (3.2.1). This gave the following block parameters for each error rate:

Pass one: $k_1 = \{70,\ 23,\ 14,\ 11,\ 8,\ 7,\ 4\}$

Pass two: $k_2 = \{302,\ 103,\ 62,\ 54,\ 42,\ 32,\ 25\}$

Pass three and four: $k_3 = 2 \times k_2$, $k_4 = 2 \times k_3$

Table 3.2.1: The results from simulation of the Cascade protocol.

$\bar{e}$	$n_{min}(\bar{e})$	$\bar{N}$	$\bar{N}_{parity}$	$[N_L, N_U]$	$[e_L, e_U]$	$[\#]Failure$
0.01	808	868	665	699, 1056	0.0076, 0.0126	48
0.03	1994	2132	1525	1899, 2387	0.0257, 0.0346	3
0.05	2864	3164	2165	2906, 3433	0.0444, 0.0558	2
0.06	3274	3649	2413	3383, 3928	0.0539, 0.0662	0
0.08	4022	4470	2801	4189, 4751	0.0729, 0.0871	1
0.10	4690	5302	3324	5016, 5600	0.0922, 0.1080	0
0.15	6098	6933	3733	6663, 7212	0.1409, 0.1593	0

Since the actual error rate in each simulation varies from the mean error rate, Figure 3.2.1 is a scatter plot which shows the actual error ratio plotted against the number of exchanged bits for all simulations.

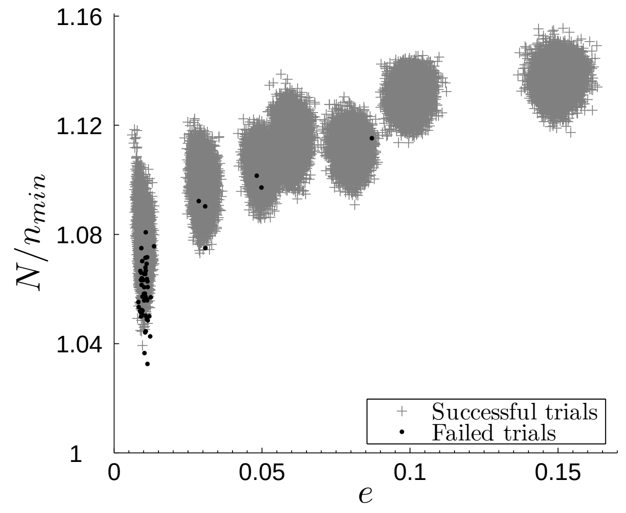


Figure 3.2.1: Ratio of information exchanged relative the lower bound set by the Shannon limit for the Cascade protocol.

I also examined how sensitive the Cascade protocol is to changes in the error rate. I simulated a total of 7000 trials with k_i's optimized for $\bar{e} = 0.05$ but used mean error rates between 0.035 and 0.065. The curve Figure 3.2.2, corresponding to Figure 3.2.1, clearly shows the expected minimum for $e = 0.05$. The curve stays below 1.16 bits above Shannon limit for the interval $e = 0.05 \pm 0.02$. As seen in Figure 3.2.2, the protocol only failed once, which matches the performance shown in Table 3.2.1.

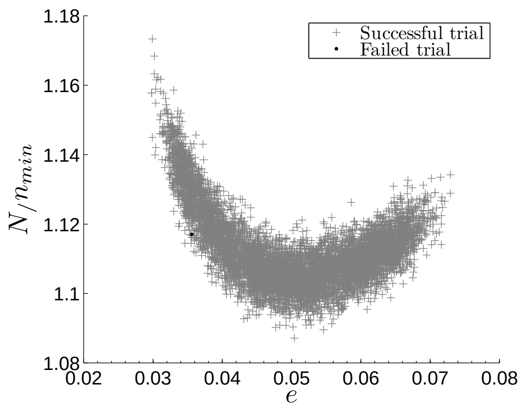


Figure 3.2.2: The ratio between exchanged parity bits and lower theoretical limit with k_i's optimized for $e = 0.05$ and varied error rate.

3.2.4 Performance Analysis

During the error-correction phase, the protocol exchanges N parity bits over the public channel. An upper bound of the information leaked to Eve will therefore be $l = N$ to be discarded in the privacy amplification. Given $n_{raw} = 100 \cdot 10^3$ bits, there will on average remain $n_{sif} = 50 \cdot 10^3$ after sifting. The empirical key generation ratio is

$$R_{Cascade} = \frac{5 \cdot (n_{sif} - N) - k - s}{n_{raw}}, \tag{3.2.2}$$

assuming $\beta = 1$ for the moment and accounting for the simulation using $n_{sif} = 10,000$ instead of the norm $n_{sif} = 50 \cdot 10^3$. To only investigate the reduction of key generation ratio due to leakage in error correction I set k and s to zero. The parameters k and s will not change the form of the graphs in Figure 3.2.3, they will only translate the graphs downwards. The simulated key generation ratio for Cascade is shown in Figure 3.2.3. The ratio is scatter plotted for each e. The Figure 3.2.3 also has the mean estimated curve for $R_{Cascade}$ together with a 99% confidence interval for both $\bar{e}$ and $R_{Cascade}$. The confidence intervals are swhon in Table 3.2.1. The theoretical upper bound is Equation (3.2.2) with N changed to n_{min}.

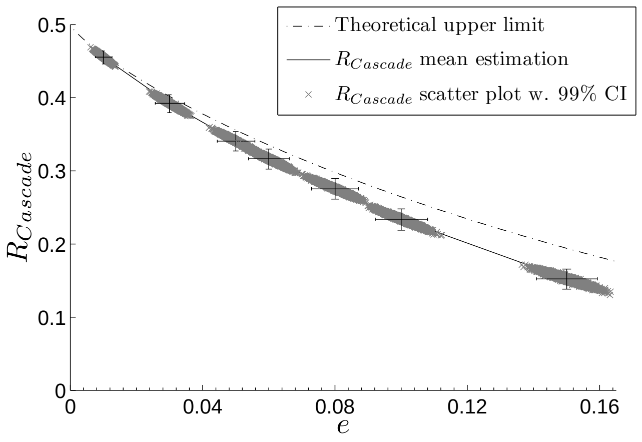


Figure 3.2.3: The estimated key generating ratio for Cascade together with the theoretical limit.

As Figure 3.2.3 shows, the Cascade protocol works close to the theoretical Shannon limit. The performance is less than 116% of the theoretical limit, see Figure 3.2.1. Therefore I have chosen $N = 1.16 \times n_{min}$ as an estimation for N. Based on this the key generation ratio for Cascade will be

$$R_{Cascade^{opt}} > \frac{[1 - 1.16 \times h(e)] \times n_{sif} - k - s}{n_{raw}} \times \beta. \tag{3.2.3}$$

The correction efficiency of the code in four passes is good for $\bar{e} \geq 0.03$. For $\bar{e} = 0.01$, the Cascade protocol failed 48 times. Using Table 3.2.1 and the theory of Section 2.2.1, Table 3.2.2 displays the following:

- y, observed number of successful trials, i.e. all errors corrected.

- $\hat{\beta}$, estimated probability of correcting all errors.
- $[\beta_L, \beta_U]$, a 99% confidence interval for the estimated $\hat{\beta}$.

Figure 3.2.4 shows $\hat{\beta}$ from Table 3.2.2 together with confidence intervals.

Table 3.2.2: The estimated probability of correcting all errors for the simulations.

$\bar{e}$	y	$\hat{\beta}$	$[\beta_L, \beta_U]$
0.01	9952	0.9952	0.9931, 0.9968
0.03	9997	0.9997	0.9989, 1.0000
0.05	9998	0.9998	0.9991, 1.0000
0.06	10,000	1.0000	0.9995, 1.0000
0.08	9999	0.9999	0.9993, 1.0000
0.10	10,000	1.0000	0.9995, 1.0000
0.15	10,000	1.0000	0.9995, 1.0000

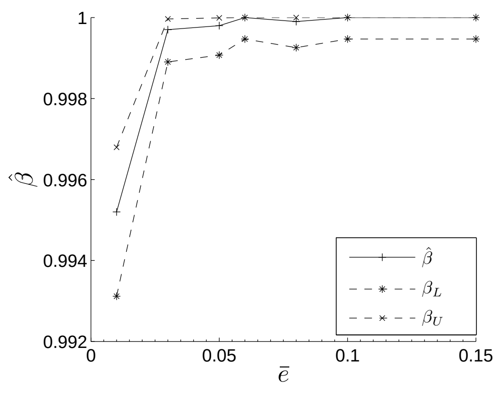


Figure 3.2.4: $\hat{\beta}$ together with 99% confidence interval.

Further investigations of the failed trials showed that a trial will fail when two erroneous bits happen to be mapped close to each other into the same block in all four passes. When they lie in the same block there will not be an odd number of errors so the algorithm proceeds to the next pass. Table 3.2.3 below shows two error bits travel through one failed trial with $\bar{e} = 0.01$. The errors had index 2672 and 2681 in Bob's original bit string.

Table 3.2.3: Two erroneous bits block-mapping in the four passes.

Pass 1		Pass 2		Pass 3		Pass 4	
Block	Index	Block	Index	Block	Index	Block	Index
38	12	30	78	3	135	2	302
38	21	30	79	3	136	2	303

As Table 3.2.3 above shows the two bits are never separated and never found in any parity check. The blocks in the different passes are about k_i bits long and thus there are approximately n_{sif}/k_i blocks in pass i. The blocks for $\bar{e} = 0.01$

are approximately 70, 302, 604 and 1208 bits long giving about 143, 33, 17 and 8 different blocks in each of the passes one to four. With $\bar{e} = 0.03$, the numbers of blocks are 435, 97, 49 and 24 making it more unlikely for the two errors to be mapped into the same block. For higher $\bar{e}$, the number of blocks in each pass increases further which makes it less and less likely for two errors to be mapped close to each other in the same block. To overcome this problem, we need to change the way of computing k_i. A lower k_i means more blocks, but it also means more communication and thus further away from the Shannon limit. Liu also proposed the settings $k_1 = \lfloor (4 \ln 2)/(3\bar{e}) \rfloor$ and $k_2 = \lfloor (4 \ln 2)/(3\bar{e}) \rfloor$ [16]. 10,000 trials with these choices for $\bar{e} = 0.01$ gave 33 failures and an average of 876 parity bits exchanged. Compare this with 48 failed trials and 868 revealed parity bits for the presented simulation.

3.2.5 Communication Aspects

In the protocol Alice and Bob exchange parity bits, N, according to previous section. Before the correction starts, they need to agree on a seed and the error rate so they use the same values of k_i. All this information can be sent as one package. The parity matrix with approximately $\frac{n_{sif}}{k_1} + \frac{n_{sif}}{k_2} + \frac{n_{sif}}{k_3} + \frac{n_{sif}}{k_4} \approx N - N_{parity}$ elements can also be sent before the interactive communication starts.

However, when Bob detects a parity mismatch when comparing the sets he needs to inform Alice which block to add to $\mathcal{K}$. Then they send parity bits in each direction to exchange parity information. This is ineffective, since they pay the cost of redundant information for each single bit. They need to exchange the parity bits for each block, thus one bit is sent in each direction. They run one binary search for each error, i.e. on average a total of $\bar{e} \times n_{sif}$ searches. The interactive communication thus consists of $\bar{e} \times n_{sif}$ block numbers and N_{parity} parity bits in the binary searches.

3.2.6 Conclusions

The Cascade protocol works very well and close to the Shannon limit, less than 116% away. It has a high probability of correcting all errors. To correct errors for $\bar{e}$=0.01, we would need lower k_i's or one more pass in the algorithm. Using the estimation of 116 % of Shannon limit would give $N = 1.16 \times n_{sif} \times h(0.01) = 937$, which is well above the simulated mean of 868, thus we can make about 70 additional parity checks and still use the estimation. Further work would be to examine how to choose k_i for the low error rates. In addition, it is better to choose a pessimistic error rate when calculation k_i's. This sacrifices closeness to Shannon limit but is less sensitive to changes in the error rate. Using the values of k_i for $e = 0.15$ to correct $e = 0.05$ will make Cascade perform at 1.5 bits above Shannon limit, compared with the presented of about 1.13 bits above. As seen in Figure 3.2.2, the protocol is not very sensitive to a change in error rate of 2%. I assume this holds for all optimized k_i's.

One drawback is the high number of interactive communicated bits forcing the classical channel to be operational for longer a time.

My implementation of Cascade was rather complicated using several different representations of the key and conversions between them. This made my implementation of Cascade perform quite slowly in correction. To be useful in a real system, the implementation needs to be done more efficient.

The Cascade protocol is one of the first protocols presented and the mostly used protocol. Therefore it will be used as norm for the comparisons of the other three protocols.

3.3 Yamamura and Ishizuka's Protocol

3.3.1 Introduction

I have chosen to abbreviate this correction protocol AYHI after the inventors, Akihiro Yamamura and Hirokazu Ishizuka [17]. This protocol uses a very different approach compared with Cascade. Alice and Bob share n_{sif} uncorrected bits. They now each divide their own string into three substrings r_1, r_2 and r_3 of lengths $q + q + u = n_{sif}$. Sacrificing r_2 and r_3, they can correct all errors in r_1 creating a key of length q. The introduction of errors into a bit string of ones and zeros can be seen as an exclusive or (xor) addition with a string of equal size where the non-zero elements represent the errors, see Figure 3.3.1 for schematic explanation. An example is

$$r_1 \oplus e_1 = \{0,\ 1,\ 1,\ 0,\ 0,\ 1\} \oplus \{0,\ 0,\ 0,\ 1,\ 0,\ 1\} = \{0,\ 1,\ 1,\ 1,\ 0,\ 0\},$$

thus introducing errors in elements four and six. The number of errors introduced is the number of non-zero elements in e_1, also known as the Hamming weight of e_1, denoted $w(e_1)$. The Hamming weight is on average the mean error rate multiplied with the length, i.e. $\bar{e} \times |e_1|$. The Hamming distance, denoted $d(r_1, r_2)$, is the number of entries that differ between two vectors of same length. In the example above $d(r_1, r_1 \oplus e_1) = 2$.

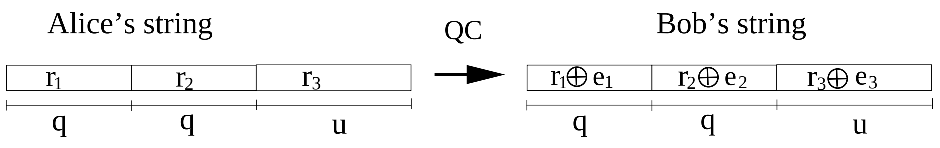


Figure 3.3.1: The fractionizing of string and introduction of errors by xor additions with the error strings.

After quantum transmission and sifting, Alice and Bob agree on a hash function

$$H : \{0,1\}^q \rightarrow \{0,1\}^u.$$

Alice sends $H(r_1)$ to Bob. Bob has $r_1 \oplus e_1$ and tries to find e_{twid} such that

$$H(r_1) = H(r_1 \oplus e_1 \oplus e_{twid}).$$

Note that $w(e_{twid} \oplus e_{twid}) = 0$, since xor addition with itself generates a vector with only zeros. If H is a good hash function, $e_{twid} = e_1$ with great probability. Unfortunately there are 2^q possibilities for e_{twid} so the method so far is simply a version of exhaustive search. There is also a security issue. If Eve can break the hash function she will have full access to r_1, i.e. the final key. To circumvent this, Alice encrypts $H(r_1)$ with r_3 using the one-time-pad technique, thus sending Bob

$H(r_1) \oplus r_3$. In order to give Bob partial information to narrow his search, Alice also sends him $r_1 \oplus r_2$, see Section 3.3.2 for further details. If H has the property of being a *locally neighborhood collision free function* [17], Bob has Equation (3.3.1) below as stopping criterion to test whether e_{twid} is correct [17]

$$d(H(r_1) \oplus e_3, H(r_1 \oplus e_1 \oplus e_{twid})) < \frac{\alpha + \bar{e}}{2} \times u. \tag{3.3.1}$$

The parameter α in Equation (3.3.1) is significantly larger than $\bar{e}$ and is decided from the properties of the hash function used. The number of non-zero elements with the same index in e_1 and e_2 is called the number of collisions, which is denoted X_c with outcome $x_c = |\{i : (e_1)_i = (e_2)_i = 1\}|$. Bob can not know the outcome x_c in advance and must test for all possible numbers. X_c is binomial distributed according to $Bin(q, e^2)$. See Section 3.3.4 for details on how this effect the number of iterations.

3.3.2 Algorithm

The AYHI algorithm is:

```
procedure AYHI (A : Alice's bit string
                B : Bob's bit string) is
begin
  – Alice and Bob agrees on $H$ and $q$, remember that $H$ gives $u$.
  – Alice send $H(r_1) \oplus r_3$ and $r_1 \oplus r_2$ to Bob.
      This is equal to one-time-pad protecting $H(r_1)$ and $r_1$.
  – Bob has $r_1 \oplus e_1$, $r_2 \oplus e_2$, $r_3 \oplus e_3$ from quantum transmission.
  – Bob calculates:
      $(r_1 \oplus r_2) \oplus (r_2 \oplus e_2) = r_1 \oplus e_2$.
      $(r_1 \oplus e_1) \oplus (r_1 \oplus e_2) = (e_1 \oplus e_2)$.
      $(H(r_1) \oplus r_3) \oplus (r_3 \oplus e_3) = H(r_1) \oplus e_3$.
  – Bob uses his information to run the following searches, aborting if $e_{twid}$ satisfies
      Equation (3.3.1).
  Assume $x_c = 0$,
      Bob can find $e_{twid} = e_1$ by exhaustive search on
      the non-zero positions in $e_1 \oplus e_2$
      and testing against Equation (3.3.1) for each guess.
      There is $2^{w(e_1 \oplus e_2)}$ different possibilities.
  Assume $x_c = 1, \ldots$
      Bob runs an exhaustive
      search by generating a start guess as if there were no collisions.
      Let $\mathcal{I}$ be the set of indices $i$ with $(e_1 \oplus e_2)_i = 0$.
      For each starting guess, Bob tests for every possible twiddle of the $x_c$ bits
      from $\mathcal{I}$.
  – If a matching $e_{twid}$ is found, Bob declares success, else the $q$ bits
      are discarded.
end AYHI
```

I have chosen to correct for up to four collisions, $x_c = 1, \ldots, 4$, based on my available computing powers.

There are many functions H with the desired property. I have chosen $sha-1 : \{0,1\}^q \to \{0,1\}^{160}$ [18], thus making $u = 160$. The property has been empirically validated [17] where the value of $\alpha = 0.25$ was stated. The $sha-1$ algorithm is a part of the *Java* standard package.

To clarify with an example

$$\begin{aligned} e_1 &= \{0,1,1,0\} \\ e_2 &= \{0,1,0,1\} \\ e_1 \oplus e_2 &= \{0,0,1,1\}, \end{aligned}$$

Note the collision in the second element, resulting in $(e_1 \oplus e_2)_2 = 0$. Bob first assumes no collisions and tries the $2^{w(e_1 \oplus e_2)} = 4$ possibilities

$$\begin{aligned} e_{twid} &= \{0,0,0,0\} \\ e_{twid} &= \{0,0,0,1\} \\ e_{twid} &= \{0,0,1,0\} \\ e_{twid} &= \{0,0,1,1\}. \end{aligned}$$

Finding no e_{twid} to satisfy Equation (3.3.1), Bob continues to try for one collision, i.e. twiddling every combination of one bit from $\mathcal{I} = \{1,2\}$ for each starting guess:

$$\begin{aligned} e_{start} = \{0,0,0,0\} &\Rightarrow \left\{ \begin{array}{lcl} e_{twid} &=& \{1,0,0,0\} \\ e_{twid} &=& \{0,1,0,0\} \end{array} \right. \\ e_{start} = \{0,0,0,1\} &\Rightarrow \left\{ \begin{array}{lcl} e_{twid} &=& \{1,0,0,1\} \\ e_{twid} &=& \{0,1,0,1\} \end{array} \right. \\ e_{start} = \{0,0,1,0\} &\Rightarrow \left\{ \begin{array}{lcl} e_{twid} &=& \{1,0,1,0\} \\ e_{twid} &=& \{0,1,1,0\} \end{array} \right. \end{aligned}$$

He aborts when finding e_{twid} that satisfies Equation (3.3.1), hence he and Alice share the key $r_1 \oplus e_1 \oplus e_{twid} = r_1$. Notice that the number of keys Bob needs to try grows exponentially in $w(e_1 \oplus e_2)$ and the number of collisions, x_c. Both these measures depend on q and $\bar{e}$. This is the prime concern when analyzing performance of AYHI. My implementation continues up till four collisions. Keys with more collisions are not corrected by my implementation.

As mentioned, there are $2^{w(e_1 \oplus e_2)}$ starting guesses. When a starting guess has been chosen, it is a combinational problem to place x_c collisions amongst the remaining $q - w(e_1 \oplus e_2)$ zeros in $e_1 \oplus e_2$. Simple combinatorics gives the maximum number of tests for each x_c as

$$N_{max}(w(e_1 \oplus e_2), x_c) = \sum_{i=0}^{x_c} 2^{w(e_1 \oplus e_2)} \times \binom{q - w(e_1 \oplus e_2)}{i}, \qquad (3.3.2)$$

where the summation comes from the algorithm trying for all possible number of collisions up to x_c.

3.3.3 Simulation Results

As mentioned above, the number of e_{twid}'s Bob needs to try grows rapidly for both longer q and larger e, the method is not feasible for $q \approx \frac{n_{sif}-u}{2}$. Alice and

Bob need to choose a reasonable large q, based on Bob's computing power and expected $\bar{e}$. They then run the protocol for

$$\left\lfloor \frac{n_{sif}}{2q+u} \right\rfloor$$

partitions. This gives key length $q_{final} = \lfloor \frac{n_{sif}}{2q+u} \rfloor \times q$ before privacy amplification. The leftover bits can either be corrected using a smaller q or be discarded.

I used the following parameters:

- $q = \{20, 30, 40, 50, 60, 70\}$
- $\bar{e} = \{0.01, 0.03, 0.05, 0.06, 0.08, 0.1\}$
- $M = 4000$
- $H = sha - 1$, giving $u = 160$ and $\alpha = 0.25$

The $\bar{e} = 0.15$ has been left out because early testing shown too long run time for practical use for this high error rate. Due to the low q, some partitions did not have any errors in them. This is discovered in the first test, i.e. with $e_{twid} = \{0, \ldots, 0\}$. Table 3.3.1 displays the result of the simulations with different values of q's. The data presented is:

- $\bar{N}$, the average number of e_{twid} Bob tested.
- $\%N < \bar{N}$, the percentage of trials where N was smaller than $\bar{N}$.
- $max(N)$, maximum number of tested e_{twid}'s for single trial.
- BER e_i, bit error rate for string r_i for the trial which gave N_{max}.
- $[\#]Failures$, number of trials where no matching e_{twid} was found.

I have chosen to regard trials with the number of collisions, x_c, equal to five as outliers and they are not included in $\bar{N}$ or $max(N)$. Each unsuccessful trial was due to $x_c = 5$, no failures was due to wrongful keys passing Equation (3.3.1). The trials where $x_c = 5$ have been included in Figure 3.3.2 below, albeit they were not successfully corrected. The simulation with $q = 70$ had to be aborted before $\bar{e} = 0.1$ was finished due to too long run time. Therefore I will not discuss the performance for $q = 70$, but I have chosen to include the Figure 3.3.2(f) for comparison.

Table 3.3.1: Simulation results for AYHI.

	$q = 20$	$q = 30$	$q = 40$	$q = 50$	$q = 60$
$\bar{N}$	21	237	$3.174 \cdot 10^3$	$6.182 \cdot 10^4$	$7.723 \cdot 10^5$
$\%N < \bar{N}$	91	94	96	97	98
$max(N)$	$1.292 \cdot 10^4$	$2.562 \cdot 10^5$	$7.132 \cdot 10^6$	$4.148 \cdot 10^8$	$2.141 \cdot 10^9$
BER e_1	0.350	0.200	0.175	0.160	0.167
BER e_2	0.250	0.233	0.250	0.140	0.217
$[\#]Failures$	0	0	1	1	1

Figure 3.3.2, on the next spread, shows N, i.e. the number of e_{twid}'s Bob tested against Equation (3.3.1) before finding a match plotted against $w(e_1 \oplus e_2)$. As mentioned, the number of test depends largely on the number of collisions, x_c, which also is shown in Figure 3.3.2. More on this in Section 3.3.4. N_{max} from Equation (3.3.2) is shown as dashed lines in the graphs in Figure 3.3.2.

The connection of N to the mean error rate and q can be seen in Figure 3.3.3 on the next spread below. Here N has been plotted in bins labeled with the error rate for the trial. $\bar{N}$ from Table 3.3.1 have been included as dashed lines in all sub figures in Figure 3.3.3.

3.3.4 Performance Analysis

I have not been able to find a good way to predict the number of e_{twid}'s that needs to be tested, N. We only have Equation (3.3.2) as an upper limit. The number of collisions is distributed according to $X_c \sim Bin(q, \bar{e}^2)$ for large enough n_{sif}. The Hamming weight of $e_1 \oplus e_2$ equals $w(e_1) + w(e_2) - 2 \times x_c$ where $w(e_1) \sim Bin(q, e)$ and $w(e_2) \sim Bin(q, \bar{e})$. Empirically $w(e_1 \oplus e_2) \sim Bin(q, 2\bar{e} - 2\bar{e}^2)$ was found to be a good fit. Figure 3.3.4(a) shows the empirical fit stated above together with simulated $w(e_1 \oplus e_2)$ for $q = 50$ and $\bar{e} = 0.08$. The simulated curve was done with 10^4 trials. Figure 3.3.4(b) shows the probability density function, pdf, for X_c with same parameters, i.e. $Bin(50, 0.08^2)$.

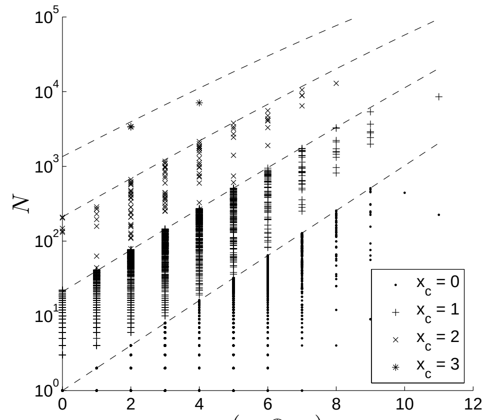
To analyze the worst case behavior of N I simulated 4000 N_{max}'s for each error rate by generating $w(e_1 \oplus e_2)$ and X_c from distributions above and inserting them into Equation (3.3.2). I used $q = 50$. Figure 3.3.5 shows the simulated upper limits as black dots together with of Figure 3.3.3(d) as gray dots. It is clearly seen that the simulations of upper limits give a good indication to the number of required tests for each error rate. For each partition, Alice sends $q + u$ bits. This is far above the Shannon limit which, at least, is less than q. This makes a poor performance when compared with the Shannon limit. For $q = 60$ and $\bar{e} = 0.1$ we have $(u + q)/(q \times h(\bar{e})) = 7$ bits above Shannon limit. Hereafter I will not make further comparisons with the Shannon limit.

The maximum number of bits discarded due to uneven partitioning is less than $2q + u$. We can consider $H(r_1)$ and r_1 one-time-pad protected using r_3 and r_2, respectively, which would be totally secure if all bits in r_1, r_2 and r_3 are truly random and unknown. However, Eve can know bits in r_2 and r_3 from eavesdropping on the quantum channel. Furthermore, Eve also knows e_1, e_2 and e_3 if we assume that Eve causes all errors. Therefore, she can decrypt some bits from r_1 in the key reconciliation phase. According to Yamamura and Ishizuka the algorithm leaks at most $l = 2 \times e \times q \approx 2 \times \bar{e} \times q$ bits per partitioning that needs be removed in the privacy-amplification step [17]. Hence, the theoretical key generation ratio will be

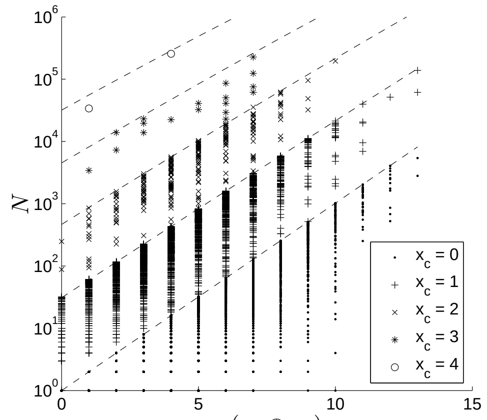
$$R_{AYHI} > \frac{\left\lfloor \frac{n_{sif}}{2q+u} \right\rfloor \times (q - 2 \times \bar{e} \times q) - (2q + u) - k - s}{n_{raw}}. \tag{3.3.3}$$

Figure 3.3.6 below shows Equation (3.3.3) with $k = s = 0$ and $u = 160$.

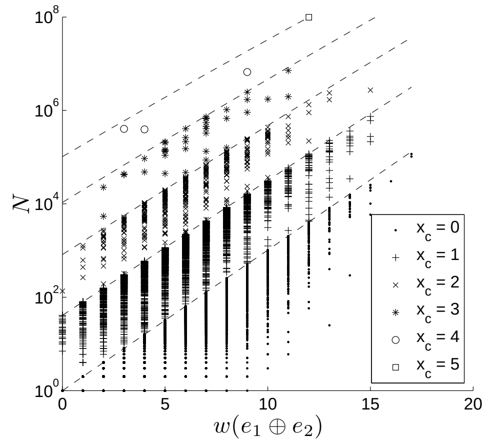
One option is for Bob to stop the search if the number of tests reaches a certain number, e.g. $\bar{N}$ from Table 3.3.1. This would mean that about 5% of substrings would be discarded, which means to multiply Equation (3.3.3) by about 0.95.



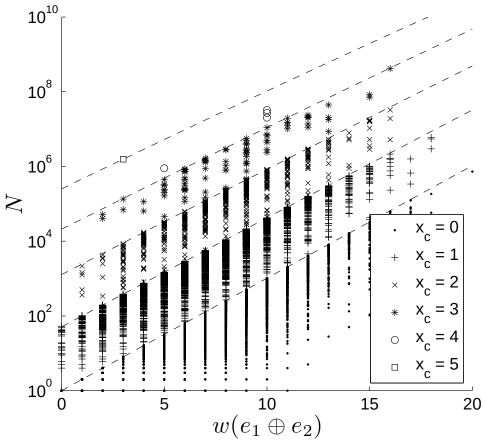
3.3.2(a): $q = 20$



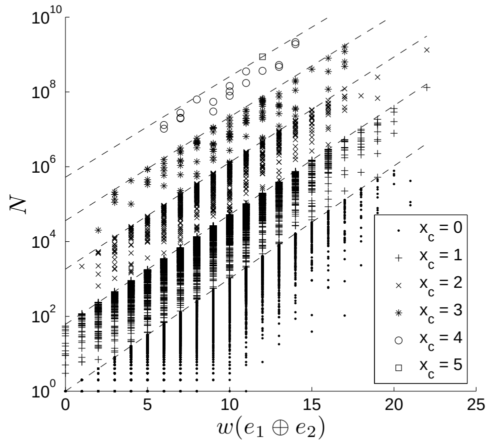
3.3.2(b): $q = 30$



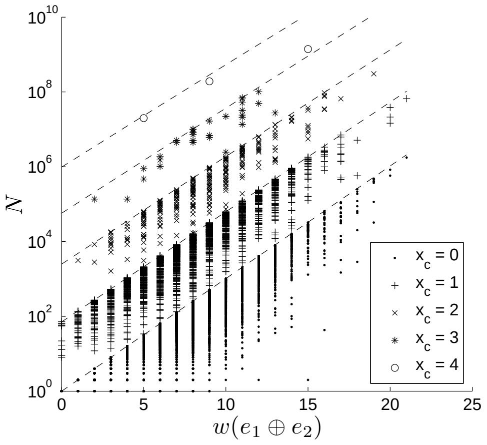
3.3.2(c): $q = 40$



3.3.2(d): $q = 50$

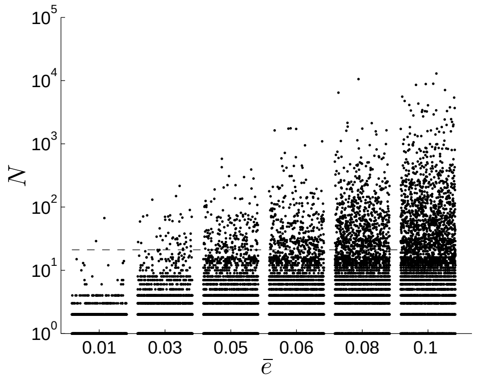


3.3.2(e): $q = 60$

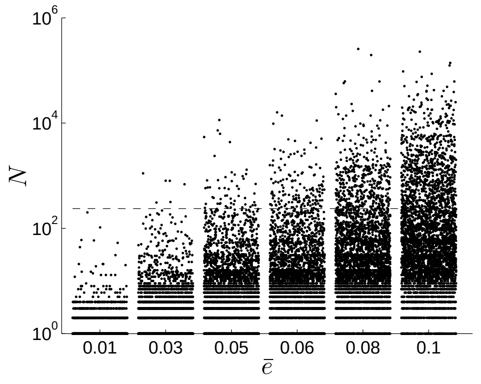


3.3.2(f): $q = 70$

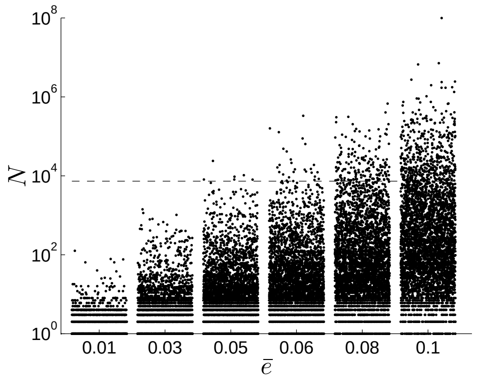
Figure 3.3.2: The number of e_{twid}'s Bob tried for each q.



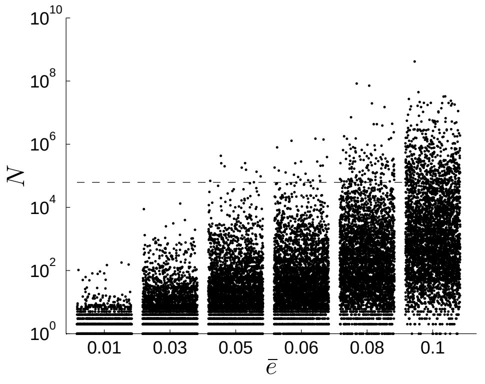
3.3.3(a): $q = 20$



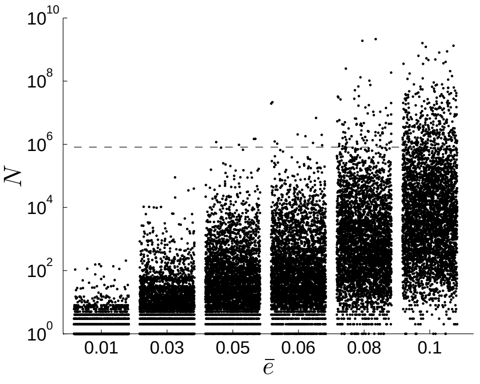
3.3.3(b): $q = 30$



3.3.3(c): $q = 40$

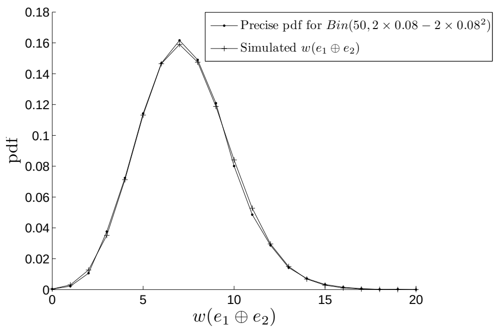


3.3.3(d): $q = 50$

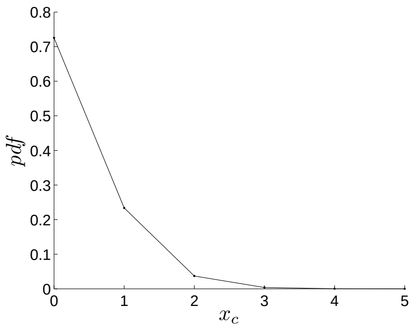


3.3.3(e): $q = 60$

Figure 3.3.3: The number of e_{twid}'s Bob tried for each q.



3.3.4(a): The probability density function of $w(e_1 \oplus e_2)$ from estimated distribution and simulation.



3.3.4(b): Pdf for $Bin(50, 0.08^2)$.

Figure 3.3.4: Distributions for $w(e_1 \oplus e_2)$ and X_c.

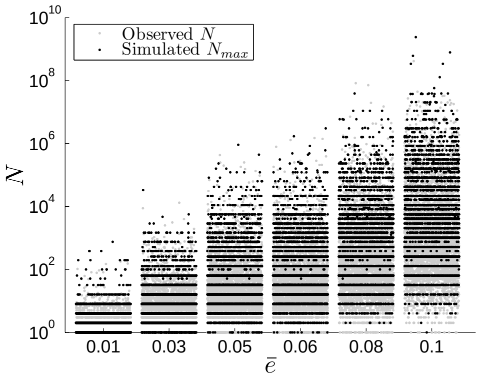


Figure 3.3.5: The simulated upper limit of tests, N_{max} together with observed number of tests for $q = 50$.

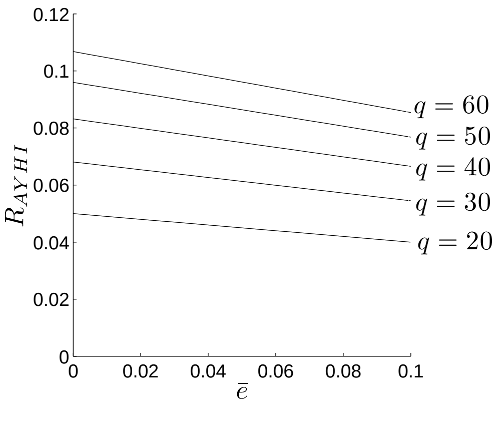


Figure 3.3.6: The key generation ratio versus $\bar{e}$.

If no wrongful key passes Equation (3.3.1), the algorithm will correct all errors up to four collisions. The probability of $x_c \leq 4$ is

$$\beta_{X_c} = P(X_c = 0) + P(X_c = 1) + P(X_c = 2) + P(X_c = 3) + P(X_c = 4). \quad (3.3.4)$$

In all trials made, no wrongful keys passed and no correct keys were missed by Equation (3.3.1). AYHI failed three times, each time due to $x_c = 5$, see Table 3.3.1. The parameter β_{Test} is the probability of the test in Equation (3.3.1) working as intended. The total β for AYHI will then be $\beta_{X_c} \times \beta_{Test}$, assuming that the number of collisions and the passing of Equation (3.3.1) are independent. The confidence interval of β_{Test} is based on 4000 trials, but it succeeded in a total of 96,000 trials. Therefore the confidence interval could be made narrower, but I have chosen to use the number of trials with the same parameters, i.e. 4000. Hence we have:

$$\hat{\beta}_{Test} = 1.0000, \quad \beta_{Test,L} = 0.9987 \text{ and } \beta_{Test,U} = 1.0000.$$

The total β_{AYHI} is therefore:

$$\beta_{AYHI} = \beta_{X_c} \times \hat{\beta}_{Test}, \quad \beta_L = 0.9987 \times \beta_{X_c} \text{ and } \beta_U = 1.0000 \times \beta_{X_c}.$$

In the confidence interval I have assumed that $P(x_c \leq 4)$ can be regarded as a constant, i.e. only a multiplicative factor.

Figure 3.3.7 shows β_{AYHI} as a function of $\bar{e}$ for different choices of q.

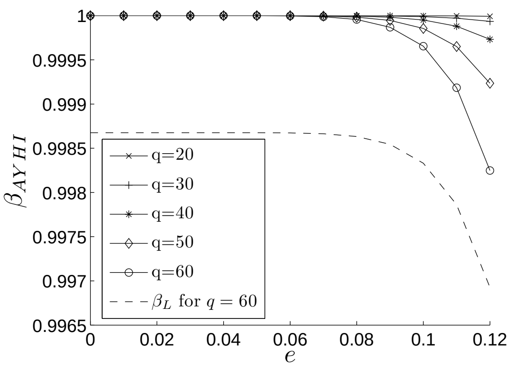


Figure 3.3.7: The probability of correcting all errors versus the mean error rate.

3.3.5 Communication Aspects

Alice and Bob need to decide on $\bar{e}$ in order to choose as low q as possible and still end up with a positive number of bits after privacy amplification. The total amount of data that needs to be transferred depends on the choice of q and u. The number of bits needed to correct the string is given by

$$N_{total} = \left\lfloor \frac{n_{sif}}{2q + u} \right\rfloor \times (q + u). \quad (3.3.5)$$

All the N_{total} bits can be sent in a large package at once. This makes the AYHI protocol much less interactive compared with the Cascade protocol. After decoding, Bob tells Alice for which fractions he found a matching e_{twid}.

3.3.6 Conclusions

The benefit of AYHI is that all this information can be send to Bob in one large block, not needing to maintain interactive communication. Bob can do his computations and afterwards tell Alice for which partitions he could find a matching key. It also has a high probability for success, especially for lower q's. The drawback of AYHI is the large amount of data needed to transfer and the need for Bob to possess large computational powers. There is a trade off between β, q, X_c and $w(e_1 \oplus e_2)$ since more collisions highly increases the number of keys to try. I implemented my algorithm in *Java* on an Intel Pentium III. The simulation for $q = 30$ took six minutes, while the simulation for $q = 60$ took over 48 hours. In my implementation the algorithm does not utilize object oriented programming. Therefore AYHI is well suited for implementation in a linear programming language such as *Fortran* which usually is faster than *Java*. Hardware implementation might also be possible. Furthermore my AYHI algorithm can easily be parted between several processors, a concept knows as parallel programming. There is no direct order in which to generate e_{twid}. Different processors can search between different possibilities, if one processor finds a matching guess the other processors are interrupted. I think an implementation in an even more low level language on a number of processors will bring the speed up enough for AYHI to be useful.

3.4 Low Density Parity-Check Code

3.4.1 Introduction

Low density parity-check code, LDPCC, was discovered by Gallager in 1962 [19], rediscovered by MacKay and Neal in the late 90s [20, 21] and finally adapted to QKD by Pearson in 2004 [22]. LDPCC as developed by MacKay and Neal was originally intended to protect transmitted data from errors by using a large sparse matrix representing different parity checks. Pearson's paper shows that it can easily be adapted to QKD by only a number of minor changes [22].

Let $\mathbf{H}$ be a very sparse binary matrix with m rows and n columns, where $m < n$ The matrix has Hamming weight t_c per column and uniformly weighted rows with approximate weight t_r. The correcting capacity of LDPCC highly depends on $\mathbf{H}$. To function properly, we need $t_c \geq 3$.

Alice has sent the binary bit string A of length n to Bob who has measured B containing a number of errors. Alice then sends Bob $\mathbf{z} = \mathbf{H}A \pmod 2$. Note that $\mathbf{z}$ will have length m. Bob uses the information in $\mathbf{z}$ and B to find the most probable $\hat{x}$ such that $\mathbf{H}\hat{x}$ (mod2) $= \mathbf{z}$. The algorithm uses probabilistic decoding and updates the probability for the value of each bit in $\hat{x}$ in each iteration. This is called belief propagation. The matrix $\mathbf{H}$ is created from a random seed shared by Alice and Bob in order for them to share $\mathbf{H}$ without sending it. Schematic of the information exchange is presented below.

$$[A] \cdot \begin{bmatrix} \mathbf{I} \\ \mathbf{H} \end{bmatrix} (\text{mod } 2) = \begin{bmatrix} A \\ \mathbf{H}A(\text{mod } 2) \end{bmatrix} \begin{array}{l} \}n \text{ bits} \rightarrow \text{Quantum Channel} \\ \}m \text{ bits} \rightarrow \text{Classical Channel} \end{array}$$

Traditional coding theory with only one noisy channel would require

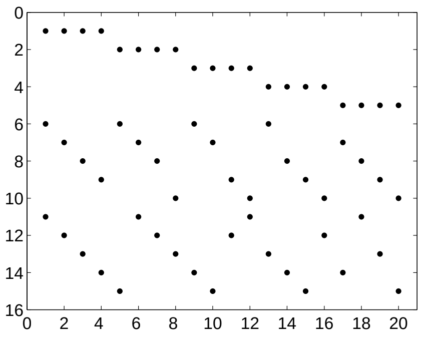
$$\frac{n}{m+n} < 1 - h(e),$$

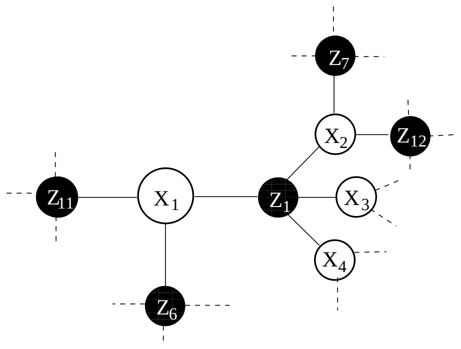
but since the parity information $\mathbf{H}A$ is sent on an approximately noiseless channel, this relation does not directly apply. The rate of the code will instead be $(n-m)/n$, thus the Shannon limit can be expressed as

$$\frac{m}{n} > h(e), \tag{3.4.1}$$

where m and n are the dimensions of $\mathbf{H}$ as before. LDPCC is considered to leak m bits to Eve, i.e. $l = m$. This gives a fraction of $1 - m/n$ bits left after privacy amplification, assuming $k = s = 0$. This requires m to be smaller than n. Otherwise more information is sent on the classical channel.

Let x_i, the element number i in $\hat{x}$, be denoted *bit* and let z_j be the j'th row of $\mathbf{H}$ multiplied with $\hat{x}$ (mod 2). The z_j's are referred to as *checks*, and is the xor addition of t_r bits. Vice versa, each x_i participates in t_c checks. To illustrate, take $\mathbf{H}$ to be the sparse matrix shown in Figure 3.4.1(a), with $m = 15$, $n = 20$, $t_c = 3$ and $t_r = 4$.


3.4.1(a): Sparse matrix $\mathbf{H}$.



3.4.1(b): Part of network of checks and bits.

Figure 3.4.1: Sparse matrix and part of belief network set by it.

According to Figure 3.4.1(a) we have $z_1 = x_1 \oplus x_2 \oplus x_3 \oplus x_4$. For example, the bit x_2 also participates in checks z_7 and z_{12}. We can build a network with the bits and checks as nodes and the arcs between them defined by the matrix $\mathbf{H}$, see Figure 3.4.1(b). In each iteration, the algorithm updates the probability for the value of each bit in $\hat{x}$. To choose the bit value, the algorithm uses the current probability of the other bits connected to it via the checks together with the probability of satisfying the connected checks. The idea is that erroneous bits will violate some checks and the algorithm chooses the bit value for the bits x_i with the highest probability. The performance highly depends on $\mathbf{H}$ being free of cycles, otherwise we can get caught in a loop. The value of a bit in decoding should not depend on its value in a previous iteration. This is related to the rank of $\mathbf{H}$. With m and n, we can only have a maximal rank of m, thus there will be cycles since $m < n$. Hopefully the belief network is big enough for decoding before the loop.

There is much work in progress finding an algorithm designing different good matrices $\mathbf{H}$ rapidly. The development is used in traditional error correction as proposed by Neal and MacKay but results are directly implementable on QKD.

3.4.2 Algorithm

MacKay and Neal give two ways of constructing **H**, based on Gaussian elimination [21]. I have not fabricated any method for building **H**, but used prefabricated matrices freely available by courtesy of David MacKay [23]. I also used the matrix generation algorithm included in the package in Reference [24]. The aim when generating matrices is to avoid short cycles in the network of x_i's and z_j's, but still be able to generate the matrices in polynomial time.

The algorithm presented by MacKay and Neal [21] has a number of variables presented and explained below. Here x is the bit value of one bit in the decoding and y_i is the output i from the BSC with error probability p, i.e. Bob's measurement. First we have

$$f_i^1 = \begin{cases} 1-p & \text{if } y_i = 1 \\ p & \text{if } y_i = 0 \end{cases}$$

and

$$f_i^0 = 1 - f_i^1.$$

These are the probabilities for the bit values of x_i based on the measurement, i.e. $P(x_i = 1|y_i) = f_i^1$. For example if we have $\bar{e} = 0.1$ and Bob measures $y_i = 1$, then $P(x_i = 1|y_i = 1,\, \bar{e} = 0.1) = f_i^1 = 0.9$.

Define the two sets I and J as $I(j) = \{i : (\mathbf{H})_{ji} = 1\}$ and $J(i) = \{j : (\mathbf{H})_{ji} = 1\}$. These are the arcs between bits and checks as specified by **H**. $I(j) \setminus i$ denotes the set $I(j)$ with the bit i excluded. The same notation is used for $J(i) \setminus j$ to exclude j's. For example, using the matrix in Figure 3.4.1(a) we have $I(j = 1) = \{1,\, 2,\, 3,\, 4\}$ and $J(i = 1) = \{11,\, 6,\, 1\}$. This is also seen in Figure 3.4.1(b).

The algorithm uses two different quantities, q_{ji}^x and r_{ji}^x, which are updated by maximum likelihood estimations in the iterations. Note that q's and r's with subscripts are variables in the algorithm and not key lengths as in the previous sections. The quantity q_{ji}^x is the probability of bit i having the value x given the information from all checks except j. Similarly the quantity r_{ji}^x is the probability of check j being satisfied if bit i is fixed at the value x and all other bits have values distributed according to their q_{ji}^x's. The final step is to calculate the final probabilities for the bit values of $\hat{x}$. These are denoted q_i^x, i.e. $q_i^1 > q_i^0 \Rightarrow \hat{x}_i = 1$. The algorithm starts with a starting distribution of q_{ji}^x and it halts if the decoding is successful or a maximum number of iterations is reached.

The tools for handling sparse matrices in MATLAB can make the implementations of this code simple, since we only need to work with the non-zero values in **H**. I based my implementation on the MATLAB code in the package by Igor Kozintsev found in Reference [24]. The package also contains a mex-file for building matrices, which I made use of without any changes to the code. The changes to the algorithm in the file `lpdc_decode.m` was

- Send **z** as input.
- Change from Gaussian channel to BSC by changing how f_i^x is calculated.
- Remove *Binary Phase Shift Keying* modulation of y_i, since we use a BSC and not a Gaussian channel.

- Insert
  `z(length(ii))=0;sPdq(find(z(ii)==1))=-sPdq(find(z(ii)==1));`
  in horizontal step to create the $(-1)^{z_j}$.

- Change stopping criterion to `mod(H*xhat,2)==z`,
  i.e checking if $\mathbf{H}\hat{x}$ (mod2) $= \mathbf{z}$.

The last two changes are given by Pearson in [22], being the adoption of normal LDPCC to QKD.

Given the notations above the LDPCC is:

```
procedure LDPCC (A : Alice's bit string
                 B : Bob's bit string) is
begin
    – Alice and Bob decide values for $m$, $n$, $t_c$, $t_r$ and a seed to create identical $\mathbf{H}$.
    – Alice sends Bob $\mathbf{z} = \mathbf{H}A$ (mod2).
    – Bob calculates $f_i^0$ and $f_i^1$ based on his measurements $y_i$
      and the estimate for the error probability of the BSC.
      Then he starts the decoding phase.
    – Initialization: Set $q_{ji}^0 = f_i^0, q_{ji}^1 = f_i^1\ \forall j$.
    do loop
        –Horizontal step: Define $\partial q_{ji} \equiv q_{ji}^1 - q_{ji}^0$ and compute for each $j, i$:

            $\partial r_{ji} = (-1)^{z_j} \prod_{i' \in I(j) \setminus i} \partial q_{ji'}$.

            Set $r_{ji}^0 = (1 + \partial r_{ji})/2$ and $r_{ji}^1 = (1 - \partial r_{ji})/2$.

        –Vertical step: For each $i$, $j$ and $x$ update:

            $q_{ji}^x = \alpha_{ji} f_i^x \prod_{j' \in J(i) \setminus j} r_{j'i}^x$,

            where $\alpha$ is chosen to normalize the probabilities so that $q_{ji}^1 + q_{ji}^0 = 1$.
            Update the 'pseudoposterior probabilities' $q_i^1$ and $q_i^0$ as:

            $q_i^x = \alpha_i f_i^x \prod_{j \in J(i)} r_{ji}^x$,

            again with $\alpha$ chosen to normalize the probabilities so that $q_i^1 + q_i^0 = 1$.

            Set $x_i = 1$ if $q_i^1 \geq 0.5$, else $x_i = 0$.

            Calculate $\mathbf{H}\hat{x}$ (mod2) and set $iterations = iterations + 1$.

    while $\mathbf{H}\hat{x}$ (mod 2) $\neq \mathbf{z}$ and $iterations \leq max_{iterations}$

    – If valid $\hat{x}$ was found in decoding, declare success and return $\hat{x}$.
        Otherwise declare failure.
end LDPCC
```

3.4.3 Simulation Results

The ability of correcting errors for this algorithm depends on the shape of **H**. I try to find the maximal correctable error-rate for each matrix by trying to correct increasingly higher error rates. Therefore I have chosen to create the errors in a different way compared with the previous simulations. Instead of making a Bernoulli test for each bit, I introduced a fix number of errors but in random positions. The number of errors introduced was $\lfloor \bar{e} \times n \rfloor$, hence the error rate is $e = \lfloor \bar{e} \times n \rfloor / n$. Then I made M trials for each error rate, thus the plots show e rather then $\bar{e}$. A trial is considered successful if a valid decoding was found which was identical to A. I denote the maximal error rate for which all trials found a valid decoding by e_{max}. This is the error rate with which closeness to the Shannon limit and probability of correction is computed. By inserting the errors this way, I will have M observations for e_{max}, and can thus make a narrow confidence interval.

The maximum number of iterations for the vertical and horizontal steps was set to be 100. This decision was based on the result in Figure 3.4.2 below, which shows the ratio of successful decodings as a function of e for different maximum number of iterations based on 100 trials for each e. Figure 3.4.2 shows no tendency towards better decoding ratio for more iterations. Based on my available computing powers, I have chosen to increase the number of trials rather than the maximum number of iterations. Albeit the MATLAB code is very compact, it is also rather slow compared with *Java*. Therefore I will only use $n \leq 20,000$, but the algorithm is of course usable for arbitrary large n. It is still worth mentioning that in some observed cases the error rate appeared to be stable at 0.1 for about 70 iterations and then suddenly drop to zero when a valid decoding was found. A $max_{iterations} = 50$ would have been too small, therefore it advisory to be careful when choosing $max_{iterations}$.

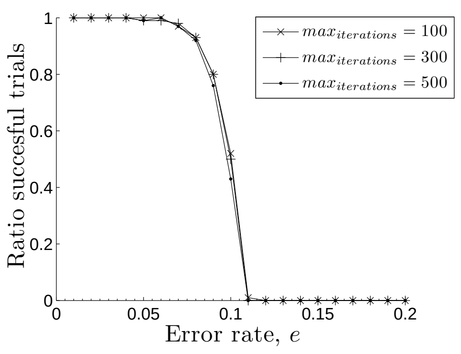


Figure 3.4.2: The rate of successful decodings as a function of e plotted for different $max_{iterations}$.

Figure 3.4.3 shows how the error rate of current $\hat{x}$ varies with the number of iterations. Typically three different patterns were found:

- The error rate dropped with each iteration until reaching zero for a valid $\hat{x}$.
- The error rate oscillates for the first iterations, but stabilizes on one value different from zero until maximum number of iterations is reached.

- The error rate drops and rises in an oscillatory fashion.

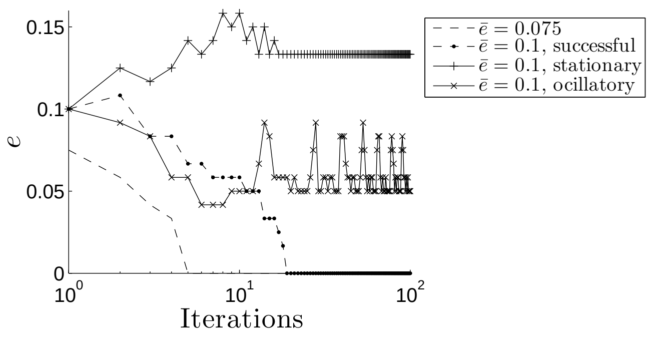


Figure 3.4.3: The error rate of $\hat{x}$ plotted against the number of iterations. Note the log scale on the x-axis.

The Figures 3.4.2 and 3.4.3 was created using a $\mathbf{H}$ with $n = 120$, $m = 64$, $t_c = 3$ and $t_r = \{5, 6\}$.

Closeness to the Shannon limit for a suitable matrix $\mathbf{H}$ is determined by the ratio between m and n, as stated in Equation (3.4.1). Therefore it is of interest to try matrices of different sizes, but with the same ratio m/n to see how the size matters.

The web source [23] contains a large variety of matrices, generated with different algorithms. Simulations revealed that matrices with same m, n, t_r and t_c generated with the same algorithm but using different seeds performed close to identical in the decoding. Therefore I will only present the result using one matrix for each m and n.

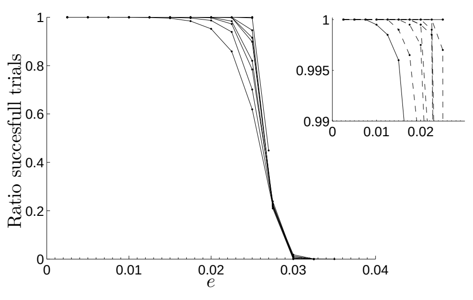
I used the following parameters common for the three simulations:

- $max_{iterations} = 100$
- $M = 2000$
- t_r as uniform as possible between the rows in the matrix.

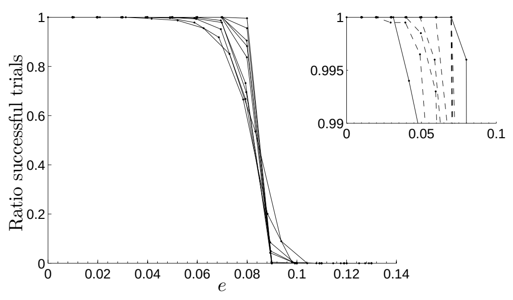
Table 3.4.1 below shows the specific parameters used. I have chosen $t_c = 5$ for $m/n = 0.25$ to compare with Pearson, who used $t_c = 5$ for a matrix with $m = 1006$ and $n = 4096$ [22]. The column "Source" refers to how the matrices were obtained. Figure 3.4.4 shows the result, with the ratios of successful trials plotted against e for each $\mathbf{H}$. The tendency is that the curves get more step-like for larger matrices. The largest matrices have an almost vertical drop to zero when e increases over the maximal error rate the algorithm can correct.

Table 3.4.1: Parameter settings for simulations of LDPCC

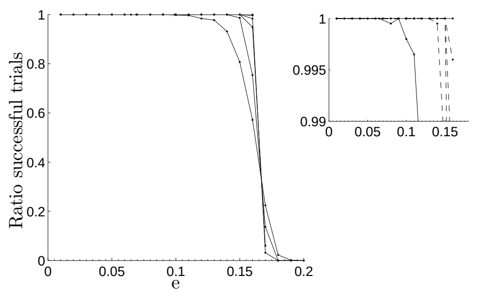
m/n	Matrices with n equal to				t_c	$\bar{e}$	Source
	400	800	1200	1600			
0.25	2400	3200	4000	12,000	5	$\{0.0025 : 0.0025 : 0.04\}$	[24]
	16,000	20,000					
	96	204	504	816			
0.50	1008	2640	4000	4896	3	$\{0.01 : 0.01 : 0.13\}$	[23]
	8000	20,000					
	200	1200	4000	6000			
0.75	10,000	18,000			3	$\{0.01 : 0.01 : 0.2\}$	[24]



3.4.4(a): $m/n = 0.25$



3.4.4(b): $m/n = 0.50$



3.4.4(c): $m/n = 0.75$

Figure 3.4.4: Ratio of successful trials plotted against e for each $\mathbf{H}$ with $m/n = \{0.25, 0.50, 0.75\}$. The small figure in each sub figure is a zoom of the larger, with the curves for the smallest and the largest matrix plotted as solid lines and the other curves as dashed.

Based on the results shown in Figure 3.4.4, I made plots of maximal correctable error-rate, e_{max}, in Figure 3.4.5. Table 3.4.2 below summarize the achieved e_{max}. The distance to the Shannon limit is calculated from Equation (3.4.1). Because $e > 0.15$ is insecure and smaller n gives faster correction, I have chosen to use $n = 4000$ and $e_{max} = 0.15$ for $m/n = 0.75$, even if it appears as if LDPCC could perform better for higher n.

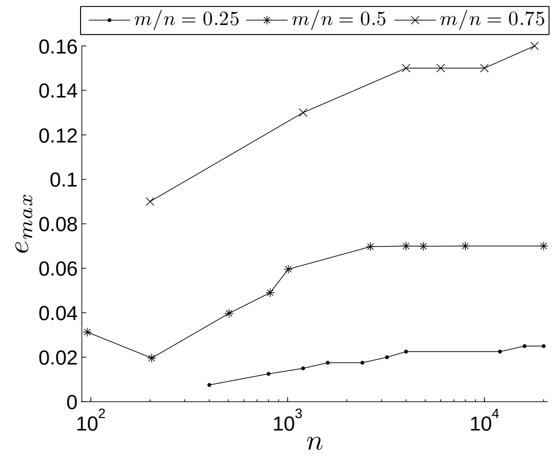


Figure 3.4.5: The e_{max} plotted against n for each $\mathbf{H}$ with $m/n = \{0.25, 0.50, 0.75\}$. Note the log scale on the x-axis.

Table 3.4.2: The e_{max} for the different m/n's.

m/n	e_{max}	$m/(n \times h(e_{max}))$ bits above Shannon limit	$n_{e_{max}}$
0.25	0.025	1.48	16,000
0.5	0.07	1.37	2640
0.75	0.15	1.23	4000

To test how the column weight, t_c, effects the error correction I did 2000 trials for twelve matrices generated from Reference [24]. The trials were conducted by creating four matrices for each m/n with $t_c = 3, \ldots, 6$ and let them correct the same random strings. I used an initial error rate, increasing it in small steps and recorded the error rate when decoding started to fail.

Table 3.4.3: The e_{max} for different t_c and m/n

m/n	$t_c = 3$	$t_c = 4$	$t_c = 5$	$t_c = 6$
1000/4000	0.012	0.022	0.022	0.020
2000/4000	0.07	0.065	0.06	0.05
3000/4000	0.15	0.12	0.11	-

Table 3.4.3 shows that Pearson's use of $t_c = 5$ was optimal for lower ratios m/n. With m/n equal to 0.5 and 0.75, $t_c = 3$ performed the best. The e_{max}'s are in agreement with the results in Table 3.4.2. Note that the matrix with $m/n = 0.75$ and $t_c = 6$ could not even correct the initial guess of $e_{max} = 0.1$.

3.4.4 Performance Analysis

Pearson reports that his implementation performed at 1.26 bits above the Shannon limit for $e = 0.03$. This is the only error rate for which he presents results. He also writes that his implementation works faster and is less computationally demanding than his implemented version of the Cascade protocol [22]. Since I have implemented my protocols in different languages, I can not comment on this. Pearson does however not mention anything about which matrices he used, how many trials he did or the estimated probability of correction.

My results are evaluations of LDPCC as a method, but also the design of the matrices used. The results in Table 3.4.2 above shows that LDPCC, with the matrices used, performs well for higher m/n. The performance is however relatively poor for $m/n = 0.25$. My implantation only got down to 1.48 bits above Shannon limit, whereas Pearson's got to 1.26. Moreover, the result where obtained with n equal to 16,000 which make the computation of both matrix and decoding time-consuming. The reason for this might be that the algorithm generating $\mathbf{H}$ does not perform well for lower m/n.

To use LDPCC in a real case, Alice and Bob first need an accurate estimate of e. One way of doing it is to compare and discard a large enough number of bits to achieve the estimate for e. Another way is to assume $e = 0.15$ and use $m/n = 0.75$ to correct a small fraction of the string. If a valid decoding was found, they have still saved 25% of the bits and can estimate e. If no valid decoding was found, they know that $e > 0.15$ and abort due to insecure error rate. Based on the estimate they agree on the m/n. A simple estimate can be made from Figure 3.4.5 and Table 3.4.2. For example, if we assume a maximal error rate of 0.09 for a real system, Alice and Bob would use $0.5 < m/n < 0.75$, e.g. $m/n = 0.65$. This gives an approximate performance of somewhere between 1.23 and 1.37 bits above Shannon limit from Table 3.4.2.

Alice and Bob can use $n = n_{sif}$, but this is slow in both matrix generation, encoding and decoding. Therefore they can create $\mathbf{H}$ with lower n and run LDPCC for n_{sif}/n blocks. I will assume they can find a n which is a factor of n_{sif}, i.e. no or very few bits fall outside the block partitioning. Bob then announces for which blocks of length n he found a valid decoding. Recall that LDPCC is considered to leak m bits, i.e. $l = m$. These need to be removed in privacy amplification.

The key generation ratio will be

$$R_{LDPCC} = \frac{\lfloor n_{sif}/n \rfloor \times [1 - m/n] \times n - k - s}{n_{raw}} \times \beta, \qquad (3.4.2)$$

where m/n comes from estimation as described above. Figure 3.4.6 shows R_{LDPCC} using the results in Table 3.4.2.

The correctional probability is good based on the simulation results. As mentioned, e_{max} is the highest error rate with 100% successful decodings for each m/n. Hence $\hat{\beta}$ is only valid for error rates smaller or equal to e_{max}. The β will be estimated to

$$\hat{\beta}_{LDPCC}(m/n) = 1.0000 \qquad [\beta_L, \beta_U] = [0.9974,\ 1.0000],$$

from 2000 trials with $m/n = \{0.25,\ 0.5,\ 0.75\}$ and error rates up to e_{max}.

There could be a question of security in LDPCC, which Pearson does not make any comments on. Since $\mathbf{H}$ is not secret, Eve could try to use her eavesdropped information to perform the decoding herself. Let us assume that Alice and Bob have estimated $e = 0.05$ and uses a pessimistic matrix with $m/n = 0.5$ to correct their strings. In an optimal intercept/resend attack as described in Section 2.3.4, this error rate gives Eve the probability 0.72 of measuring the correct outcome. The effective error rate of Eve's string will thus be 0.28. Fortunately for Alice

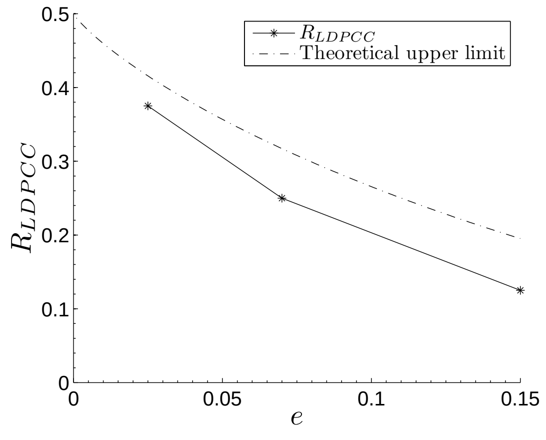


Figure 3.4.6: The empirical key generation ratio for LDPCC together with theoretical upper limit set by the Shannon limit.

and Bob this decoding would require $m/n > h(0.28) = 0.86 > 0.5$. Hence Eve has to little information to decode, as she is on the wrong side of the Shannon limit. However, I have not included other attacks such as for example PNS. This issue needs a more careful analysis. We need to examine whether there is a possibility of designing an eavesdropping scheme, including all attacks, capable of gathering enough information for decoding. Alice and Bob should therefore choose matrices as close to the Shannon limit as possible, thus leaving Eve little room for decoding. If a risk of security is found, Alice and Bob need to keep their matrices secret. This would ruin the method, since we then use $m \times n$ secret bits to correct n. If they only keep the seed for the matrix secret, the security of the final key will only be as strong as the security of the seed, i.e. we gain no secret bits.

If Bob fails to decode, he could request the physical value of one key bit from Alice. Using this bit, Bob can set the value f_i^x to one or zero and try to decode again. This can be repeated until decoding succeeds. Of course they have to add one to l for every bit, but they still save key bits instead of discarding. This does however introduce interactive communication, which we would like to avoid.

In a real system, Alice and Bob will of course have detailed and validated tables of which m and n to use given an estimation of e. My MATLAB implementation was rather slow, but Pearson reports that LDPCC is faster than his version of Cascade. This is more suitable in a high-speed continuously transmitting system, where the speed of error correction is of more importance than the remaining fraction of the transmitted string.

3.4.5 Communication Aspects

The LDPCC protocol has very little interactive communication. After Alice and Bob have decided on the seed for $\mathbf{H}$, Alice splits her strings into blocks of length n and sends Bob $[n_{sif}/n] \times m$ bits. All this information can be sent before Bob starts his decoding. After decoding, Bob can declare for which blocks he found a valid decoding. This gives a total of

$$N_{total} \approx n_{sif} \times m/n. \qquad (3.4.3)$$

Of course Alice and Bob need some interactive communication to change **H** to match current $\bar{e}$. This is however only exchanged at specific times, e.g. once every thirty seconds, and will thus be a small part compared with the correctional information.

3.4.6 Conclusions

LDPCC has the large benefit of low interactive communication and low N_{total}. It performs relatively close to the Shannon limit, between 150% and 123% above it in my implementation. The drawback is that the generation time of matrices **H** grows as n^3. The encoding and decoding time also increases, therefore one should use as low m and n as possible. Another drawback is the fact that once m/n has been chosen we will fix the performance relative to the Shannon limit. If they have made a very pessimistic assumption, they will sacrifice unnecessary bits, whereas for example the Cascade protocol has a more adaptive fixation. Therefore the actual error rate should be closely monitored and new matrices fabricated adaptively.

The work in algorithms for matrix generation in classical error correction with sparse matrices might further push down this limit. This is known as finding good codes. This means finding high performance matrices. Such results are directly applicable on QKD error correction with LDPCC.

Further work also includes the careful analysis of how different attacks may give Eve enough information of perform decoding.

3.5 Winnow Protocol

3.5.1 Introduction

The Winnow protocol was presented by Buttler *et al.* in 2003 [25]. The protocol works similar to the Cascade protocol, but instead of a binary search it uses a different technique which lowers the amount of the interactive communication required. Another contrast to Cascade is that the Winnow protocol discards bits to maintain privacy. Hence we can set $l = 0$ and only use k and s in privacy-amplification phase.

As the Cascade protocol, the Winnow protocol works with parities of smaller blocks and random permutation of the strings between passes. The blocks are of length $N = 2^m$ with $m \geq 3$. The difference is that it uses a Hamming hash-function instead of binary search. This hash function is represented by a binary matrix, compressing each block into *syndromes*, see Section 3.5.2 for details.

By comparing the xor difference in the syndromes between Alice's and Bob's block, errors can be detected. With only one error in the block, the syndrome difference gives the binary representation of the position of the error. To correct it, Alice flips this bit. This strategy only works for one error. In the case of two errors, flipping the bit given by the syndrome difference will induce a third error. For three errors, there is a chance of either creating a fourth error, correcting one or not effect the errors at all. For the Winnow protocol to work, the probability of single error in blocks needs to be greater than the probability of two or more. This is however the case for Alice's and Bob's strings.

To avoid the case with even errors, Alice and Bob exchange the parities of the blocks and only run the Winnow protocol for the blocks with parity mismatch. To maintain privacy, Alice and Bob discard one bit from the blocks with no parity mismatch and $\log(N) + 1 = m + 1$ bits from blocks with parity mismatch.

The Winnow protocol needs to be run in several passes in order to achieve an acceptable final error rate, e.g. $e_{final} \leq 10^{-6}$. The length of the blocks in the different passes determines how much the error rate is reduced and how many bits are discarded. A shorter block handles higher error rates, but discards more bits.

3.5.2 Algorithm

After the quantum transfer, Alice has the string A and Bob the string B. They part their strings into blocks of size $N = 2^m$ with $m \geq 3$. I denote a block taken from B by the length of the block as a sub-index, e.g. B_8 is a block of length 8 from B. The first bit in each block will be discarded to maintain privacy in th parity check, so the effective length of the blocks in Winnow protocol will be $N_h = 2^m - 1$. My implementation discarded the remaining bits after block fractionalizing, thus creating $\lfloor n_{sif}/2^m \rfloor$ blocks. Empirical trials below revealed that this had only minor effect on the key generation ratio.

The Hamming hash function is denoted $h^{(m)} \in \{0,1\}^{m \times N_h}$. It is calculated as

$$h_{i,j}^{(m)} = \left\lfloor \frac{j}{2^{i-1}} \right\rfloor \pmod 2, \tag{3.5.1}$$

where $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, N_h\}$. The matrix $h^{(m)}$ transposed gives the binary representation of the numbers $1, \ldots, N_h$ as the rows. The matrix for $m = 3$ is:

$$h^{(3)} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

For each block with parity mismatch, Bob calculates the syndrome $S_b = h^{(m)} \cdot B_{N_h}$ and sends it to Alice. Remember that the fist bit in each block has been discarded after the parity check. He also sends the number of the block, so that Alice know which blocks to correct. Alice then calculates the *syndrome difference* $S_d = S_a \oplus S_b \in \{0,1\}^m$, where S_a is her syndrome corresponding to S_b. If there are non-zero elements in S_d, i.e. $w(S_d) > 0$, Alice flips the bit in A_{N_h} given by $S_{d,1} \cdot 2^0 + \ldots + S_{d,m} \cdot 2^m$. $S_{d,i}$ is the bit i in the syndrome S_d. If there was only one error in the block, this will now be corrected. If there for example were three errors, there now will be two or four errors. After the bit-flip, they discard the m bits at positions 2^j with $j \in \{0, \ldots, m-1\}$. Erroneous bits might be discarded as well. There is a possibility of starting with eight bits with three errors, introducing a fourth and then discard four bits including one error, leaving four bits with three errors.

The Winnow protocol is:

```
procedure Winnow (A : Alice's bit string
                  B : Bob's bit string) is
begin
    – Agree on the number of passes and $m$ for each pass,
      based on the estimation of $e$.
    – Agree on a seed for the random permutations for each pass.
    for For all passes loop
        – Apply random permutation to strings.
        – Part their strings into blocks of size $N = 2^m$,
          which gives $N_h = 2^m - 1$.
        – For each block, Alice sends her parities to Bob.
        – For each block, Bob calculates the parities. If
          parities mismatch between his and Alice's,
          he calculates the syndromes $S_b = h^{(m)} \cdot B_{N_h} \pmod 2$
          and sends all syndromes to Alice together with block numbers.
          This way, Alice will know for which
          blocks parities matched, and the syndrome of the remaining
          blocks.
        – Then run:
        for All blocks loop
            – Bob does:
            if Parities match then
                – Discard only first bit.
            else
                – Discard bits 1 and $1 + 2^j$ with $j \in \{0, \dots, m-1\}$ from current block.
            end if

            – Alice does:
            if Parities match then
                – Discard fist bit.
            else
                – Discard first bit.
                – Calculate $S_d = S_a \oplus S_b$.
                if $w(S_d) > 0$ then
                    – Flip the block bit given by $S_{d,1} \cdot 2^0 + \ldots + S_{d,m} \cdot 2^m$.
                end if

                – Discard bits $2^j$ with $j \in \{0, \dots, m-1\}$ from current block.
                  Remember that these bit corresponds to bits $1 + 2^j$
                  in original block, since the first bit has already been discarded.
            end if

        end loop

    end loop

end Winnow
```

Section 3.5.4 describes how to choose the number of passes and m for each pass in order to achieve acceptable low e_{final} and maintain as long string as possible. This decision is based on the error rate. They also need to randomly permute their strings before each pass prior to parting into new blocks. This is to separate errors. Of course, Alice and Bob can send all required information before starting to correct in each pass. Alice sends her parities for all blocks in current pass and Bob replies with syndromes for blocks with parity mismatch. Then Alice runs the Winnow protocol and Bob discards one or $m + 1$ bits based on the parity checks. To illustrate an example with $m = 3$. Let Alice's and Bob's block be

$$\begin{aligned} A_8 &= \{1\,0\,0\,0\,0\,0\,0\,0\} \\ B_8 &= \{1\,0\,0\,1\,0\,0\,0\,0\}. \end{aligned}$$

A discarded bit is denoted '$\cdot$', which does not count in length or index. It has been included for comparison only. This gives

$$\begin{aligned} A_7 &= \{\cdot\,0\,0\,0\,0\,0\,0\,0\} \\ B_7 &= \{\cdot\,0\,0\,1\,0\,0\,0\,0\} \end{aligned}$$

after discarding first the bit. A pass of Winnow gives the syndromes

$$\begin{aligned} S_a &= \{0\,0\,0\} \\ S_b &= \{1\,1\,0\} \\ S_d &= \{1\,1\,0\}. \end{aligned}$$

The error is thus in the bit $2^0 + 2^1 = 3$ in A_7 which is correct! Alice flips this bit. After this, Alice and Bob discards bits 1, 2 and 4 from A_7 and B_7 resulting in

$$\begin{aligned} A_4 &= \{\cdot\ \cdot\ \cdot\ 1\ \cdot\ \ 0\,0\,0\} \\ B_4 &= \{\cdot\ \cdot\ \cdot\ 1\ \cdot\ \ 0\,0\,0\}. \end{aligned}$$

Each pass will hopefully lower the error rate until it get below chosen acceptable e_{final}.

3.5.3 Simulation Results

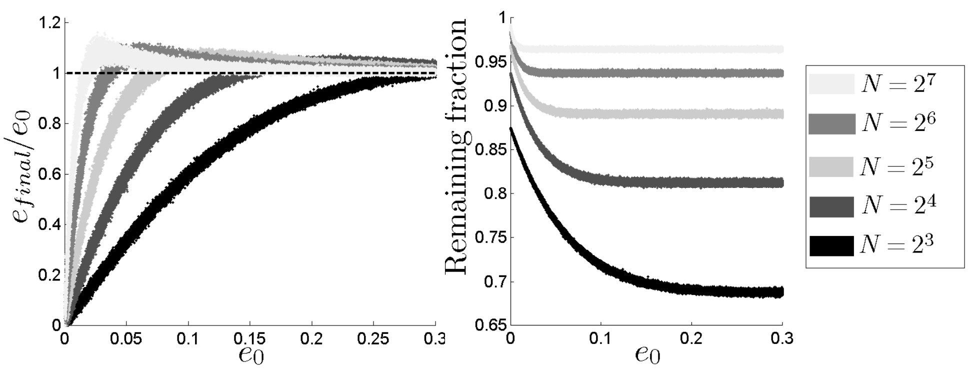
As mentioned above, the Winnow protocol needs to be run in a number of passes. In order to investigate the performance of Winnow, I simulated 10^5 single-pass Winnow (SPW) trials. I used the starting error rate, e_0, uniformly distributed in the interval $e_0 \in [0, 0.3]$. Figure 3.5.1 below shows both the ratio e_{final}/e_0 and η, the remaining fraction of the string, for each m after a SPW. The data is then used to find good combinations of passes and m's.

The expected fraction η for a SPW with given m and e_0 is

$$\eta_m = \frac{2^m - 1 - m \times \sum_{n_i=1,\ odd}^{2^m} \binom{2^m}{n_i} e_0^{n_i} (1-e_0)^{2^m - n_i}}{2^m}, \tag{3.5.2}$$

according to [25]. The total remaining fraction η after $\{j_3 j_4 j_5 j_6 j_7\}$ passes will then be

$$\eta = \prod_{m=3}^{7} \prod_{1}^{j_m} \eta_m. \tag{3.5.3}$$



3.5.1(a): Error correction ratio.

3.5.1(b): Fraction of string remaining.

Figure 3.5.1: The empirical performance for a single pass of Winnow (SPW).

The notation

$$\{j_3 j_4 j_5 j_6 j_7\} \rightarrow \eta$$

means j_3 passes with $m = 3$ followed by j_4 passes with $m = 4$ and so on, which results in η fraction of bits remaining. One has to update e_0 before evaluating the next η_m, treating each SPW independently, receiving input from the previous one.

Using the mean values of the curves in Figure 3.5.1(a) for e_{final}/e_0 and Equation (3.5.3) for η, I set up an optimization problem to solve; Given an e_0 and desired e_{final}, choose the number of passes and the corresponding m's that maximizes the remaining fraction and satisfies e_{final}. Based on available computing powers, I set the maximum number of passes for each m to five.

Table 3.5.1 shows the result of the optimization. It should be pointed out that the final error rates are based on the average of the curves in Figure 3.5.1(a) as mentioned. For very low error-rates, the empirical curves displayed a non-smooth behavior. This behavior was not included in the average estimations, which were extrapolated into $[0, 0]$. The actual e_{final} may therefore not agree with predicted. Further empirical investigation will reveal the true case. The expected η is based on Equation (3.5.3) above.

Table 3.5.1: Predicted good choices of passes and m's.

	$e_{final} < 10^{-6}$	$e_{final} < 10^{-9}$	$e_{final} < 10^{-12}$
$e_0 = 0.01$	$\{00112\} \rightarrow 0.89$	$\{00105\} \rightarrow 0.88$	$\{00024\} \rightarrow 0.88$
$e_0 = 0.03$	$\{01104\} \rightarrow 0.75$	$\{01115\} \rightarrow 0.74$	$\{01123\} \rightarrow 0.73$
$e_0 = 0.05$	$\{10122\} \rightarrow 0.64$	$\{10203\} \rightarrow 0.64$	$\{10124\} \rightarrow 0.63$
$e_0 = 0.06$	$\{11022\} \rightarrow 0.60$	$\{11015\} \rightarrow 0.59$	$\{11024\} \rightarrow 0.58$
$e_0 = 0.08$	$\{20112\} \rightarrow 0.51$	$\{20105\} \rightarrow 0.50$	$\{20114\} \rightarrow 0.50$
$e_0 = 0.10$	$\{21022\} \rightarrow 0.43$	$\{21015\} \rightarrow 0.43$	$\{21024\} \rightarrow 0.42$
$e_0 = 0.15$	$\{40112\} \rightarrow 0.25$	$\{40023\} \rightarrow 0.25$	$\{40114\} \rightarrow 0.25$

To examine the performance of the Winnow protocol I used the following settings in simulations:

- $n_{sif} = 50,000$ bits
- $M = 10,000$
- $e_0 = \{0.01,\ 0.03,\ 0.05,\ 0.06,\ 0.08,\ 0.10,\ 0.15\}$
- $\{j_3 j_4 j_5 j_6 j_7\}$ from columns in Table 3.5.1.

Tables 3.5.2-3.5.4 displays the following results of the simulation.

- $\bar{e}_{final}$, the mean final error rate of all trials.
- $[\# \, Failures]$, the number of trials with $e_{final} > 0$.
- $\bar{\eta}$, the mean remaining fraction of all trials.
- Approximate 99% confidence interval for η.

My implemented Winnow protocol ran faster than Cascade, therefore I used a five times greater n_{sif} in simulations. This made the spread in error rate around $\bar{e}_0$ smaller. Because of this, I will not include a confidence interval for e_0.

Table 3.5.2: Empirical results for desired $e_{final} < 10^{-6}$.

	$\bar{e}_{final}$	$[\# \, Failures]$	$\bar{\eta}$	$[\eta_L, \eta_U]$
$\bar{e}_0 = 0.01$	$1.997 \cdot 10^{-7}$	43	0.883	$[0.876, 0.889]$
$\bar{e}_0 = 0.03$	$1.490 \cdot 10^{-7}$	27	0.744	$[0.730, 0.754]$
$\bar{e}_0 = 0.05$	$5.106 \cdot 10^{-7}$	72	0.637	$[0.624, 0.648]$
$\bar{e}_0 = 0.06$	$9.323 \cdot 10^{-7}$	134	0.592	$[0.579, 0.606]$
$\bar{e}_0 = 0.08$	$1.187 \cdot 10^{-6}$	142	0.503	$[0.492, 0.513]$
$\bar{e}_0 = 0.10$	$1.161 \cdot 10^{-6}$	109	0.426	$[0.411, 0.437]$
$\bar{e}_0 = 0.15$	$2.066 \cdot 10^{-6}$	112	0.250	$[0.238, 0.261]$

Table 3.5.3: Empirical results for desired $e_{final} < 10^{-9}$.

	$\bar{e}_{final}$	$[\# \, Failures]$	$\bar{\eta}$	$[\eta_L, \eta_U]$
$\bar{e}_0 = 0.01$	0.0	0	0.872	$[0.864, 0.879]$
$\bar{e}_0 = 0.03$	0.0	0	0.726	$[0.714, 0.737]$
$\bar{e}_0 = 0.05$	$1.651 \cdot 10^{-7}$	26	0.635	$[0.622, 0.645]$
$\bar{e}_0 = 0.06$	0.0	0	0.584	$[0.569, 0.597]$
$\bar{e}_0 = 0.08$	0.0	0	0.495	$[0.483, 0.508]$
$\bar{e}_0 = 0.10$	0.0	0	0.417	$[0.404, 0.429]$
$\bar{e}_0 = 0.15$	$3.153 \cdot 10^{-6}$	117	0.248	$[0.233, 0.261]$

Table 3.5.4: Empirical results for desired $e_{final} < 10^{-12}$.

	$\bar{e}_{final}$	$[\# \, Failures]$	$\bar{\eta}$	$[\eta_L, \eta_U]$
$\bar{e}_0 = 0.01$	0.0	0	0.871	$[0.861, 0.881]$
$\bar{e}_0 = 0.03$	0.0	0	0.728	$[0.719, 0.739]$
$\bar{e}_0 = 0.05$	0.0	0	0.626	$[0.612, 0.638]$
$\bar{e}_0 = 0.06$	0.0	0	0.582	$[0.569, 0.594]$
$\bar{e}_0 = 0.08$	0.0	0	0.494	$[0.483, 0.506]$
$\bar{e}_0 = 0.10$	0.0	0	0.416	$[0.401, 0.429]$
$\bar{e}_0 = 0.15$	0.0	0	0.245	$[0.234, 0.257]$

3.5.4 Performance Analysis

The empirical results for η in Table 3.5.2 to 3.5.4 are in good agreement with the predicted from Table 3.5.1. The tendency for empirical fractions to be somewhat lower then predicted could be caused by discarding bits left outside even block partitions. From the results shown in Table 3.5.3, it is clear that the predicted combination of passes did not match reality and Winnow performed badly for $\bar{e} = 0.05$ and $\bar{e} = 0.15$.

The key generation ratio of the Winnow protocol is

$$R_{Winnow} = \frac{n_{sif} \times \eta - k - s}{n_{raw}} \times \beta. \tag{3.5.4}$$

Remember that Winnow preserves the privacy in the key reconciliation by discarding bits, thus we have $l = 0$. Assuming $\beta = 1$ and $k = s = 0$, the empirical curve based on the simulation results for $e_{final} < 10^{-6}$ is shown in Figure 3.5.2. The corresponding curve for $e_{final} < 10^{-12}$ is almost identical to Figure 3.5.2. The curve includes a 99% confidence interval for both e_0 and R.

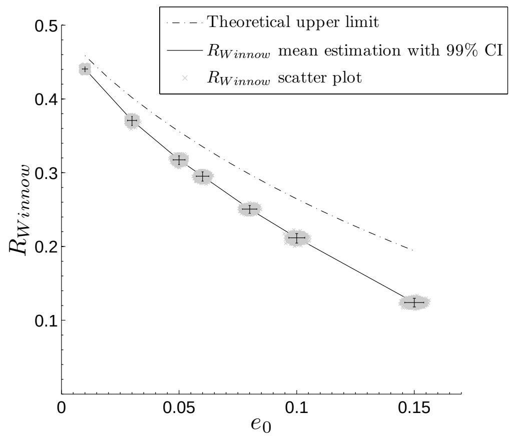


Figure 3.5.2: The empirical key generation ratio for Winnow together with theoretical upper limit set by the Shannon limit.

To convert a final error rate into β we need to do some thinking. Errors may remain after correction if two erroneous bits were mapped together in all passes, thus making the parity comparison useless. This will cause an even number of remaining errors. Errors may also come from running the Winnow protocol on block with more than one error. Assume that one block has three errors, all in positions which will be discarded. The syndrome difference could make Alice flip a correct bit, thus giving a fourth error. When bits are discarded, there will remain one error in the block. If this happens in the last pass, this error will remain. Simulations showed that over 98 % of failed trials had an even number of remaining errors. There were two remaining errors in over 90% of the failed trials. I will therefore assume a failed trial has two errors. Thus for given e_{final}, we need on average $2/e_{final}$ bits to get two errors. These bits can be parted into $(2/e_{final})/(n_{sif} \times \eta)$ final keys of length $n_{sif} \times \eta$. This gives the average fraction of failed trials as

$$\frac{e_{final} \times n_{sif} \times \eta}{2}. \tag{3.5.5}$$

Table 3.5.5 shows the expected (E) number of failed trials, i.e. Equation (3.5.5) multiplied with 10^4 using $e_{final} = \{10^{-6}, 10^{-9}, 10^{-12}\}$ together with the observed (O) from Tables 3.5.2-3.5.4. The observed number differs from expected because the actual $\bar{e}_{final}$ archived differed from desired e_{final}. If $\bar{e}_{final,archived}/\bar{e}_{final,desired} = 0.25$, we have a four times lower observed value.

Table 3.5.5: Observed number (O) of failed trials with expected number (E).

	$e_{final} < 10^{-6}$		$e_{final} < 10^{-9}$		$e_{final} < 10^{-12}$	
e_0	O	E	O	E	O	E
0.01	43	220	0	0.22	0	$2.2 \cdot 10^{-4}$
0.03	27	186	0	0.18	0	$1.8 \cdot 10^{-4}$
0.05	72	159	26	0.16	0	$1.6 \cdot 10^{-4}$
0.06	134	148	0	0.15	0	$1.5 \cdot 10^{-4}$
0.08	142	126	0	0.12	0	$1.2 \cdot 10^{-4}$
0.10	109	106	0	0.10	0	$1.0 \cdot 10^{-4}$
0.15	112	63	117	0.06	0	$0.6 \cdot 10^{-4}$

Based on Tables 3.5.2-3.5.4 and 3.5.5, we can draw the conclusion that our prediction for $e_{final} < 10^{-12}$ was good. The empirical results matched our predicted performance. We could also use the prediction for $e_{final} < 10^{-9}$ except for $e_0 = 0.05$ and $e_0 = 0.15$. The design of passes for $e_{final} < 10^{-12}$ is however better. It sacrifices insignificantly more bits and has a few more passes, which cause some more communication, but the gain in correctional capability is substantial. We can correct a 1,000 times longer string with $e_{final} < 10^{-12}$ and get the same expected count of failed trials as for $e_{final} < 10^{-9}$.

I calculate $\hat{\beta}$ from Table 3.5.5 above. The probability β represents both expected number of successes from Equation (3.5.5) and the probability to achieve the desired final error rate. Since Winnow performed equally well for all e_0, I only present one $\hat{\beta}$ based on 10,000 trials:

$$\hat{\beta}_{Winnow} = 1.0000 \qquad [\beta_U, \beta_L] = [0.9995,\ 1.0000].$$

3.5.5 Communication Aspects

The Winnow protocol is interactive, but it only exchanges information at the start of each pass. The problem in analyzing the amount of information is that the length of the string is shortened in each pass. This makes it hard to know how many blocks there will be in the next pass. However, both Alice and Bob know before each pass how many blocks they will use. Alice then sends Bob the parities for each block. Bob responds with the syndromes for which parities did not match together with which blocks the syndromes belong to. The communication before each pass thus consists of only two packages. The number of passes is $\sum_{m=3}^{7} j_m$. Hence Alice and Bob will at least know before correction how many packages will be send. The Winnow protocol has still much less interactive communication than the Cascade protocol. Figure 3.5.3 below shows the number of parity bits sent by Alice and the number of syndrome bits returned by Bob for ten trials with $\bar{e} = 0.15$ and $e_{final} < 10^{-12}$. The number of syndrome bits for each pass is the number of syndromes times the current m. Figure 3.5.3 does not include the block numbers also sent by Bob.

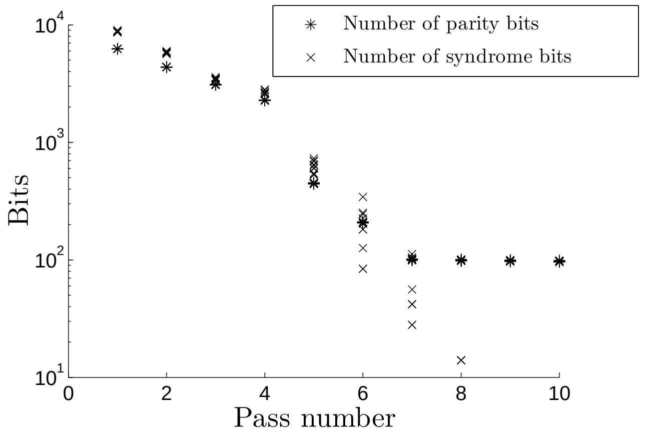


Figure 3.5.3: The number of bits exchanged for the Winnow protocol with $\bar{e}_0 = 0.15$.

3.5.6 Conclusions

The Winnow protocol performs similar to the Cascade protocol, but has the advantage of less interactivity due the use of syndromes. This eliminates the need for the highly interactive binary searches. After privacy amplification, the key generation ratio is not very far behind the one of Cascade.

Winnow also has a high probability of successful correction, apparently independent of the error rate. The correctional capability is however related to the estimate of e. If we underestimate e, Winnow will not succeed in correction due to wrong setting of $\{j_3 j_4 j_5 j_6 j_7\}$. On the other hand, if we overestimate e, Winnow will correct the errors but discard unnecessary many bits. A small test showed that $\{j_3 j_4 j_5 j_6 j_7\}$ for $\bar{e} = 0.15$ from third column i Table 3.5.1 corrected all errors for actual error rate $\bar{e} = 0.05$ but left a fraction of 0.44 instead of 0.63. Further testing revealed that a change in error rate of 1% had only minor effect on the fraction remaining and Winnow still corrected all errors.

Moreover, Winnow was easy to implement and correction trials ran very fast compared with Cascade.

Further work should include examinations of $m > 7$ and a better estimation of e_{final}/e_0 for low error rates. Moreover, more passes than five with each m may also improve the performance.

Chapter 4

Performance Comparison

4.1 Introduction

In this chapter I will compare the performance of the four error-correction methods with each other. The comparison will be done for three different error rates; 0.03, 0.05 and 0.08. I will discuss all steps, from the sifted uncorrected strings to the final secure key after privacy amplification.

I will assume $n_{raw} = 100 \cdot 10^3$ bits per second and $\mu = 0.1$ for a real life system. This gives $n_{sif} = 50$ kbit/s. All through the rest of this chapter I will assume a one second time span, and thus omit "bits per second". The results still holds for a slower transfer speed, one only has to wait longer until $n_{sif} = 50,000$ has been received. Furthermore, we can assume that Alice and Bob share enough initial secret information to authenticate the classical transmission.

Before correction starts, Alice and Bob randomly permute their strings to use the BSC model. After this they disclose a number of bits to estimate e. I have chosen to disclose 10%, i.e. the first 5,000 bits. They can choose the first bits, since their strings have been shuffled already. This gives a new n_{sif} of 45,000 bits. A better way to estimate e would be to assume a high e and correct a small part of the string, and then count the number of corrected bits, and then adaptively change the estimation of e. For simplicity I assume they estimate e the by sacrificing bits. Remember that Alice and Bob will abort if e is to high for QKD to be considered secure.

After the error correction, there is a probability of having remaining errors. Recall that we compress the string by the use of a hash function in privacy amplification. If we compress the string as one whole, one remaining error would ruin the entire string. To avoid this, we can part our string into substrings at the cost of s bits per substring. In this case one remaining error would only ruin part of the string. I will use two substrings. To validate the key after privacy amplification, Alice and Bob must sacrifice further bits. If there was an error remaining, this will now be multiplied due to the hash function and thus it will be easy to discover if the strings did not match. The validation can be done by comparing random bits and then discarding them. If any of the compared bits did not agree, the entire string is discarded. Testing revealed that 3% is enough to sacrifice after privacy amplification in order to detect string mismatch of only one bit for $n_{sif} = 10,000$. I arbitrarily set the desired level of security to $I(K;V) < 10^{-30}$

bits, which gives $s = 101$ from Equation (2.3.1).

I will assume that Eve performs the optimal individual intercept/resend attack from Section 2.3.4 on single-photon pulses and a photon-number-splitting (PNS) attack on the multiple-photon pulses. With $\mu = 0.1$, 95% of the n_{sif} bits will come from single-photon pulses. Thus Eve will know 5% of n_{sif} from PNS and additional bits from intercept/resend attack on the remaining 95% of n_{sif}. In total Eve will learn

$$k = n_{sif} \times \{0.05 + 0.95 \times \left[1 - h\left(\frac{1+\sin\theta}{2}\right)\right]\} \text{ bits,} \tag{4.1.1}$$

by eavesdropping the quantum channel. Eve can choose θ and cause $e = (1 - \cos\theta)/2$ from her attack. This estimation may be very far from reality, but I will use it to compare the correctional methods. Better models for k will only numerically change the calculations. Error rates of 0.03, 0.05 and 0.08 gives

$$\begin{aligned} k_{0.03} &= 0.1314 \times n_{sif} \\ k_{0.05} &= 0.1847 \times n_{sif} \\ k_{0.08} &= 0.2630 \times n_{sif} \end{aligned}$$

4.2 Final Key Lengths and Correctional Probability

Common assumption is that Alice and Bob has $n_{sif} = 45,000$ and a good estimate of the error rate, e. Further more I use k_e as above and security parameter $s = 202$, since we part the corrected string in two before privacy amplification and thus loose 101 bits per part. Due to the 3% sacrifice to validate the key, all equations are multiplied with 0.97. Recall the notation β for the probability of correcting all errors.

Protocol specific assumptions are:

Cascade

The final length of the key after all steps is

$$r \approx 0.97 \times \left[(1 - 1.16 \times h(e)) \times n_{sif} - k_e - s\right].$$

The $\hat{\beta}$ presented in Table 3.2.3 used $n_{sif} = 10,000$. To translate $\hat{\beta}$ to fit 45,000 bits, I regroup the simulations. Regard 10,000 trials with 10,000 bits as 2222 trials with 45,000 bits, thus groping 4.5 trials into one. The new $\hat{\beta}$ will then be out of 2222 trials. For example with $e = 0.03$, Cascade failed three out of 10,000 trials. I conservatively assume that the three failed trials will ruin three separate new trials, i.e. three out of 2222 trials.

Yamamura and Ishizuka

As in Section 3.3, I assume that the hash function is $sha-1$. This gives $u = 160$. AYHI does not need an estimation of e, but it helps in choosing block lengths q based on Bob's available computing powers.

AYHI parts the n_{sif} bits into $\lfloor n_{sif}/(2 \times q + u) \rfloor$ fractions of length q. The final key length is

$$r = 0.97 \times \left[\left\lfloor \frac{n_{sif}}{2 \times q + u} \right\rfloor \times (q - 2 \times e \times q) - k_e - s \right]$$

The drawback is the low key generation ratio, see Figure 3.3.6. The k_e will translate the graphs downwards, even below zero for shown the q's in the graph. To have a positive R_{AYHI}, we need larger q's. The minimal for positive number of final key bits together with chosen q's, respectively are

$$\begin{array}{lcl} q_{0.03} \geq 33 & \rightarrow & q_{0.03} = 100 \\ q_{0.05} \geq 59 & \rightarrow & q_{0.05} = 100 \\ q_{0.08} \geq 142 & \rightarrow & q_{0.08} = 150 \end{array}$$

Unfortunately the number of test would be too high for the two higher error rates. See Section 3.3 for explanation of the tests. An e of 0.03, $q = 100$ was computable in under a minute with my implementation. The case for $e = 0.05$ with $q = 100$ and $e = 0.08$ and $q = 110$ was too hard for a near real-time application. The expectation value of the upper limit N_{max} from Equation (3.3.2) is

q	$E\{N_{max}\}$
$q_{0.03} = 100$	$12.5 \cdot 10^3$
$q_{0.05} = 100$	$61.2 \cdot 10^5$
$q_{0.08} = 110$	$3.99 \cdot 10^{14}$

If the number of tests, N, is about 1% of the expected maximal value, then for $q = 150$ Bob has to perform approximately $4 \cdot 10^2$ tests for each of the $\lfloor n_{sif}/(2 \times 150 + 160) \rfloor \approx 100$ fractions. This gives about $4 \cdot 10^{14}$ tests which is too time consuming.

Since I have not been able to perform enough simulations to create trials for $n_{sif} = 45,000$, I can not make good estimations on $\hat{\beta}_{Test}$. Based on previous calculations I assume $\hat{\beta}_{Test} = 1.0$ for all q's. I present the total probability $\beta = \hat{\beta}_{Test} \times \beta_{X_c}$ from Equation (3.3.4) with the q's from above.

Low Density Parity-Check Codes

The correctional capability is determined by the dimensions m and n of the sparse matrix $\mathbf{H}$. The curves in Figure 3.4.5 levels out above $n \approx 4000$. We want to keep as small n as possible to reduce the encoding, decoding en matrix-generation time. Therefore I assume $4000 < n < 10,000$. The problem is to choose appropriate corresponding m's. I assume the deviation from e is no more than 30%, i.e. $e_{max} \leq 1.3 \times e$.

With $n_{sif} = 45,000$ I choose $n = \lfloor 45,000/7 \rfloor = 6428$ which only leaves 4 bits outside blocks. This gives the following m/n and m from linear interpolation between the simulation results for e_{max} versus m/n:

$$m/n_{0.03} \approx 0.345 \qquad m/n_{0.05} \approx 0.476 \qquad m/n_{0.08} \approx 0.606$$

The final key length will be

$$r = 0.97 \times [n_{sif} \times (1 - m/n_e) - k_e - s] .$$

The final key lengths appear low, but this is due to a pessimistic estimation of e_{max}. As seen in Table 3.4.2, we got to 1.37 bits above Shannon limit for $e_{max} = 0.07$. Further and finer testing with more ratios m/n will give material for better estimations, thus less pessimistic limits. A more stable error rate in the quantum transfer will also help the estimation of e_{max}.

Based on simulation results in Section 3.4, I assume that LDPCC with $n = 6428$ and m's as above will succeed in the same number of trials as the simulations presented, i.e. 2000. This corresponds to 286 trials with $n_{sif} = 45,000$. The confidence interval will thus be quite broad due to small number of trials.

Winnow

I will use the passes described by $\{j_3 j_4 j_5 j_6 j_7\}$ from the third column in Table 3.5.1, which corresponds to $e_{final} < 10^{-12}$. This was concluded to be the best choice of passes. From Table 3.5.4, we then get the remaining fraction of the key, η, for the different error rates.

We have

$$r = 0.97 \times [n_{sif} \times \eta - k_e - s],$$

Since I made 10,000 trials with $n_{sif} = 50,000$, I simply restate $\hat{\beta}$ from Section 3.5.

Summary

Using all the assumptions above, the resulting final key length are given in Table 4.2.1 and converted to key generation ratio in Figure 4.2.1. The corresponding correctional capability β together with confidence interval is shown in Table 4.2.2.

Table 4.2.1: Final key length in bits for the different protocols.

	Cascade	AYHI	LDPCC	Winnow
$r_{0.03}$	$28.0 \cdot 10^3$	$5.4 \cdot 10^3$	$22.7 \cdot 10^3$	$25.8 \cdot 10^3$
$r_{0.05}$	$21.1 \cdot 10^3$	$2.65 \cdot 10^3$	$14.6 \cdot 10^3$	$19.1 \cdot 10^3$
$r_{0.08}$	$11.8 \cdot 10^3$	180	$5.5 \cdot 10^3$	$9.9 \cdot 10^3$

Table 4.2.2: $\hat{\beta}$ for the different protocols.

	Cascade	AYHI	LDPCC	Winnow
$\hat{\beta}_{0.03}$	0.9986	1.0000	1.0000	1.0000
$\beta_{L,U}$	[0.9951, 0.9998]	[-, -]	[0.9816, 1.0000]	[0.9995, 1.0000]
$\hat{\beta}_{0.05}$	0.9991	1.0000	1.0000	1.0000
$\beta_{L,U}$	[0.9958, 1.0000]	[-, -]	[0.9816, 1.0000]	[0.9995, 1.0000]
$\hat{\beta}_{0.08}$	0.9995	0.9992	1.0000	1.0000
$\beta_{L,U}$	[0.9967, 1.0000]	[-, -]	[0.9816, 1.0000]	[0.9995, 1.0000]

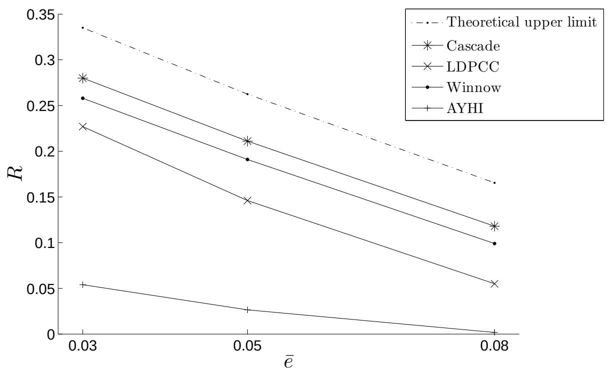


Figure 4.2.1: The final key generation ratio for the four protocols together with the upper limit set by the Shannon limit.

4.3 Communication

All protocols require an initial package of information to set the different parameters. This information is however comparatively small and approximately equal for all protocols, hence I will not include it in the comparison. The hash function H_3 from Section 2.3.3 used in privacy amplification is implemented as a random binary matrix of size $r \times n_{sif}$. This needs to be truly random and thus generated by one part and the send over the public channel. This matrix is also common for all protocols and thus I will not include it either.

I will use the error rate of 5%, since it is in the middle and I only want to present the rough estimate of information sent.

Protocol specific assumptions for this comparison are:

Cascade

Bob needs to inform Alice of the blocks for which parities did not match. This is done before the correction in each pass starts, i.e. some integers. I will not include this information because it is comparatively small. Remember that Alice and Bob need to exchange parity bits, i.e. one parity bit in each direction.

The presented numbers is 4.5 times the numbers for $\bar{e} = 0.05$ shown in Table 3.2.1.

Yamamura and Ishizuka

I will use the q's stated in the section above.

Low Density Parity-Check Codes

I will use the m/n stated in the section above.

Winnow

I will use the upper limit

$$\frac{n_{sif}}{2^m} \times j_m \times (m+1),$$

for each pass, thus ignoring the shortening of the string from bit-discarding. This is the number of blocks times the number of passes with this j_m times the m syndrome bits and one parity bit. Furthermore, Bob does not send the syndrome bits when parities did match, whereas the estimation always includes the syndrome bits. This limit gives a rough estimate. I have left out the block number, since it is not send for all blocks. The number of passes is $\sum_{m=3}^{7} j_m$, which is about ten. The empirical trials shown in Figure 3.5.3 gave about 38,000 transferred bits, which is lower than my estimate for $e = 0.05$ shown below.

Summary

Table 4.3.1 shows the estimated number of bits needed to transfer.

Table 4.3.1: Rough estimates of the number of bits transmitted in correction.

	Cascade	AYHI	LDPCC	Winnow
Initial bits	4500	32500	21400	-
Bits in one interactive round	2	-	-	$< 22500 - < 2800$
Number of round trips	9900	-	-	8
Total number of bits	24300	32500	21400	< 52700

Chapter 5

Conclusions and Further Work

In this thesis, I have performed fairly extensive simulations of four different implemented error-correction protocols intended to work in a free-space Quantum Key Distribution set-up. The classical channel is intended as an optical transmission which will work in the same path as the quantum channel. As mentioned there are three important measures for which the correction protocols have been tested. They are: length of final secure key, correctional probability and amount and type of communication required. The length of the final secure key is related to closeness to Shannon limit in the sense that we reveal as little information as possible. First I shortly present the results for the four protocols :

The Cascade protocol has been around for a decade and will be used as norm in the comparison. It performs close to the theoretical Shannon limit, only at most 1.16 bits above. This gives it the longest remaining key of the four protocols. The cost for this is the very high level of interactive communication due to binary searches. The correctional probability is good for error rates around three percent and above. For lower error rates, around one percent, my implementation failed too many times for Cascade to be considered a good choice.

Yamamura and Ishizuka's protocol AYHI looked promising in the paper, but simulations revealed that it is more of a theoretical mathematical result, rather than a useful error-correction protocol. The problem lies in the low key generation ratio. The performance depends on chosen block length, but the computations required for positive number of bits after error correction and privacy amplification is too large. It has however a low degree of interactivity and high probability of successful correction. Moreover, we do not need an estimate of $\bar{e}$ for error correction, other than to set a good block length. To save the idea, we need more powerful computers. If we could use block lengths of a thousand bits, AYHI is an option to be considered.

The Low Density Parity-Check Code (LDPCC) works very well. Albeit the implementation does not perform close the Shannon limit, at most 1.5 bits above, the low amount of interactive communication is very desirable. Given correct choice of ratio between the dimensions of the sparse matrix, the LDPCC code also has an high correctional capability. However, the LDPCC is sensitive to changes in error rate. The other protocols can correct various error rates without changing the settings. LDPCC needs new matrices if the error rate suddenly grows above assumed maximal correctable error rate, e_{max}. There are still some factors that need to be analyzed further, but LDPCC looks very promising.

Finally, the Winnow protocol, which works as something between LDPCC and Cascade. It uses blocks and parities as Cascade, but sparse matrices to perform searches as LDPCC. Since it discards bits to preserve privacy, its performance is far from the Shannon limit. However, after privacy amplification, it still performs almost identical to the Cascade protocol. The correctional probability is the best one of the four protocols. The communication is still interactive, but more of the one-way type. Alice sends Bob the parities of all her blocks, and Bob responds with his parities and his syndromes. These are considerably larger packages than just one information bit. Another benefit with Winnow was that it was easy to implement and fast in correction.

The conclusion is that Cascade, LDPCC and Winnow all have individual characteristics that makes them good choices for error correction in different situations.

If the foremost desire is to preserve key length, i.e. the quantum channel has considerably lower capacity then the classical, Cascade is the best option. The drawback is that we then need a reliable and fast high capacity classical channel. In an optical fibre setup, the classical channel fill these requirements and thus Cascade will work well. However, the Cascade protocol's interactivity will cause one side to halt in correction whilst the other is calculating. The LDPCC has the advantage of low interactive communication and fast decoding, since correctional information can be sent by Alice at the same time as Bob is decoding previous bits. This makes it good for a continuously running system, such as an optical fibre communications link. It is possible that all fibre communications in the future will have a parallel QKD system securing the information.

In the free-space setup between mobile units, we can expect that we need to preserve as many key bits as possible. The classical communication will have good capacity whilst operational, but is more unreliably than a fibre channel. The Winnow protocol will be the best choice here, since we have a good number of key bits left and controlled communication. Even if it is interactive, the information is only send at fixed intervals. The LDPCC protocol also has a low amount of interactivity. The drawback is that the amount of remaining key bits is about half the amount of Winnow and Cascade for my implementation. LDPCC is also very sensitive to changes in the error rate. However, there are single matrices presented which perform very good. Progress in the development of algorithms for the generation of such matrices is directly applicable on QKD-LDPCC. Therefore it could outperform the other two protocols with more research. A failure in decoding can be saved by sending additional bits forcing interactive communication, but the total interactive communication will still be less than for Cascade.

All three protocols are dependent of the error rate when choosing parameters. This makes it important to carefully monitor the error rate and adapt accordingly. The Cascade protocol was shown to be robust to changes of 2%. This will cause communication between Alice and Bob which this thesis has not focused on commenting. The aim was the information relevant for correction of two given binary strings.

Further work consists of two parts, one to perform bigger simulations and finer testing of the correctional protocols and the other one to actually implement the protocols into an experimental set-up.

Over all, larger simulations should be performed to give a firmer statistical conclusion of the performance of the protocols. The thesis has not focused on optimizing the computational performance of the different protocols. The number of iterations was monitored for AYHI and LDPCC and the simulations ran faster for Winnow than the other protocols, but otherwise no information was gathered on memory usage or number of operations. A more careful implementation optimizing performance for each protocol together with the counting of operations could reveal new findings. These could lead to different conclusions on which protocol to use in various scenarios.

An implementation using all four protocols would be interesting to examine. For example, use LDPCC to correct initial part of the string to estimate the error rate and still save bits and then adaptively change between protocols to achieve optimal performance.

The Cascade protocol has a lot of potential in examining how different block partitioning effects the performance of the protocol. For lower error rates, the implementation needs more work help the low correctional capability.

The LDPCC protocol needs the most work before it becomes operational. First it would be advisory to implement it in some faster language, since MATLAB is slow compared with other languages. Some articles propose hardware implemented matrix-generation algorithms, which would be faster than software implementations. Furthermore, the design of good matrices is a big research field on its own. All the results are directly implementable in the QKD error-correction protocol. The further tests of LDPCC would be to examine the maximal correctable error-rate as a function of the shape and size of the sparse matrices used. The goal would be to generate a complete 2D surface for e_{max}. Finally for LDPCC, the possibility of Eve being able to achieve successful decoding should be carefully analyzed. The protocol will be cumbersome if we need to keep the matrices secret. Then we need to sacrifice key bits to generate the next matrix and share more initial secret data than the bits we need for authentication.

The Winnow protocol should be examined with block lengths larger than 2^7 and more than five passes with each block length. A better analysis of the performance of a single Winnow pass could give a better estimate to use when choosing the number and order of block lengths. The Winnow protocol's setting of passes is dependent of the error rate. Sudden changes in the error rate require changing block lengths. Therefore simulations with wrong choice of passes should be performed to see how Winnow handles undetected sudden changes in error rate. Simple testing indicated that settings for higher error rates corrected lower error rates but with larger amount of discarded key bits.

The four protocols presented in this thesis are of course not the only ones used in QKD. Many of these are based on the Cascade protocol, but with different parameters optimized or different techniques instead of binary search. Further work would be to study these other protocols in the same way as this thesis has examined its protocols.

Literature gives examples of other ways of privacy amplification. Liu states that privacy amplification can also be performed with *extractors*, based on a small number of truly random bits [16]. Extractors use asymptotically smaller fraction of bits from the partially secure string. These should also be studied and compared with performing the privacy amplification step by using Carter and Wegman's H_3 from [11] as described in Section 2.3.3.

It would be very interesting to fit the correction protocols into an experimental free-space optical communication channel. The design of good communication protocols for implemented error-correcting codes is the next step. This can be done without the quantum channel. Alice and Bob simply agree on a binary bit string each with expected error rate between them, and use the optical classical channel to correct it. This protocol should include all steps of key reconciliation including authentication, validation of key and adaptation to changing error rate. Once such a system is build, realistic measurements on the communication required can be made and new conclusions can be drawn.

The next step after this would be to also implement the quantum channel. I firmly believe that the error-correction protocols will perform equally well on a simulated binary string with given error rate as one exchanged through the quantum channel.

Although there are many obstacles to overcome in QKD, the dream of Earth-Satellite systems with free-space QKD is realistic with enough funding for research. In such a system, more work will surely be done on authentication and error correction. This thesis has only nibbled at this great field of research!

Bibliography

[1] Rivest R. L., Shamir A., and Adleman L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[2] Singh S. *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography.* London : Fourth Estate, cop., 1999. ISBN: 1-85702-879-1, (From `www.libris.kb.se`).

[3] Bennett C. H. and Brassard G. Quantum cryptography: Public key distribution and coin tossing. *International Conference on Computers, Systems & Signal Processing*, pages 175–179, 1984.

[4] Bennett C. H., Bessette F., Brassard G., Savail L., and Smolin J. Experimental quantum cryptology. *Journal of Cryptology*, 5:3–28, 1992.

[5] Gisin N., Ribordy G., Tittel W., and Zbinden H. Quantum cryptology. *Reviews of Modern Physics*, 74:145–195, 2002.

[6] Shannon C. E. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948.

[7] Lütkenhaus N. Estimates for practical quantum cryptology. *Physical Review A*, 59:3310–3319, 1999.

[8] Gisin N. and Massar S. Optimal quantum cloing machines. *Physical Review Letters*, 79:2153–2156, 1997.

[9] Hughes R., Nordholt J., Derkacs D., and Peterson C. Practical free-space quantum key distribution over 10 km in daylight and at night. *New Journal of Physics*, 4:43.1–43.14, 2002.

[10] Bennett C. H., Brassard G., and Robert J-M. Privacy amplification by public discussion. *SIAM Journal of Computing*, 17:210–229, 1988.

[11] Carter J. L. and Wegman M. N. Universal classes of hash functions. *SIAM Journal of Computing*, 18:143–154, 1979.

[12] Cederlöf J. Authentication in quantum key growing. Master's thesis, Applied Mathematcs, Linköpings Universitet, 2005. LiTH-MAT-EX- -05/18 - - SE.

[13] Inamori H., Lüteknhaus N., and Mayers D.
Unconditional security of practical quantum key distribition. `http://arxiv.org/abs/quant-ph/0107017`, (Verified 2005-05-27).

[14] Kurtsiefer C., Zarda P., Halder M., Gorman P. M., Tapster P. R., Rarity J. G., and Weinfurter H. Long distance free space quantum cryptology. *Quantum Optics in Computing and Communications, Proceedings of SPIE*, 4917:25–31, 2002.

[15] Brassard G. and Salvail L. Secret-key reconciliation by public discussion. *Advances in Cryptology- EUROCRYPT'93, Lecture Notes in Computer Science*, 765:410–423, 1994.

[16] Liu S. *Information-Theoretic Secret Key Agreement.* PhD thesis, Techische Universiteit Eidhoven, 2002.

[17] Yamamura A. and Ishizuka H. Error detection and authentication in quantum key distribution. *Lecture Notes in Computer Science*, 2119:260–273, 2001.

[18] Federal Information Processing Standards Publications. Secure hash standard. `http://www.itl.nist.gov/fipspubs/fip180-1.htm`, (Verified 2005-08-23).

[19] Gallager R. G. Low-density parity-check codes. *IRE Transactions on Information Theory*, IT-8:21–28, 1962.

[20] MacKay D. Good error-correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory*, 45:399–431, 1999.

[21] MacKay D. and Neal R. M. Near shannon limit performance of low density parity check codes. *Electronic Letters*, 33:457–458, 1997.

[22] Pearson D. High-speed QKD reconciliation using forward error correction. *The Seventh International Conference on Quantum Communication, Measurement and Computing*, pages 299–302, 2004.

[23] MacKay D.
Encyclopedia of sparse graph codes.
`http://www.inference.phy.cam.ac.uk/mackay/codes/data.html`, (Verified 2005-08-23).

[24] Kozintsev I.
Matlab programms for encoding and decoding of ldpc codes over gf(2^m).(600 kb). `http://www.kozintsev.net/soft/ldpc_distr.zip`, (Verified 2005-08-23).

[25] Buttler W. T., Lamoreaux S. K., Torgersson G. H., Nickel G. H., Donahue C. H., and Peterson C. G. Fast, efficient error reconciliation for quantum crytology. *Physical Review A*, 67:52303–1–8, 2003.

Appendix A

Notations

$\oplus$, xor	:	Exclusive or, $x_i \oplus y_i = x_i + y_i \pmod 2$.
$\lvert \cdot \rvert$	:	Absolute value if $\cdot$ is a scalar, number of elements if $\cdot$ is a vector or a set.
$\lfloor \cdot \rfloor$	:	Highest integer still less or equal to the number $\cdot$.
$\lceil \cdot \rceil$	:	Lowest integer still greater or equal to the number $\cdot$.
$w(e)$	:	Hamming weight of e, the number of non-zero elements in e.
$d(e_1, e_2)$	:	Hamming distance, the number elements that differ between e_1 and e_2.
n_{raw}	:	Quantum bits received by Bob.
n_{sif}	:	Number of bits remaining after sifting.
e	:	Bit error rate, i.e. number of errors divided by number of bits in string.
$\bar{e}$	:	Mean bit error rate.
n_{min}	:	The lower Shannon limit on number of bits.
N_{total}	:	The total number of bits transferred in error correction.
BER	:	Bit Error Rate.
β	:	Probability of the error-correction protocol successfully correcting all errors.
$\bar{\epsilon}$	:	Disturbance parameter.
l	:	Number of bits leaked in error correction.
k	:	Number of bits leaked on the quantum channel.
s	:	Safety parameter in privacy amplification.
r	:	Length of final key.
u	:	Length out output from hash function in AYHI.
q	:	Partitioning length in AYHI.
p_{dc}	:	Probability of dark count in Bob's detector.
x, y	:	Bit strings, usually of same length, often information sent through some channel.
x, y	:	Observations of stochastic variables X and Y.
m, n	:	$\mathbf{H} \in \{0,1\}^{m \times n}$ in LDPCC.
m	:	Parameter determining block length in Winnow protocol.

A	:	Alice's bit string, i.e. Alice's key.
B	:	Bob's bit string, i.e. Bob's key.
E	:	Eve's bit string, i.e. Eve's guess of A or B.
A_i	:	Bit i in Alice's bit string.
B_i	:	Bit i in Bob's bit string.
M	:	Number of trials for each parameter setting in simulation.
R	:	Key generation ratio, r/n_{raw}.
K	:	The final secure key.
V	:	Eves gathered knowledge of $e, e(x), f, f(x)$ and g.
X, Y	:	Stochastic variables
η	:	Fraction of key remaining in Winnow.
$\mathbf{H}$	:	Sparse matrix used in LDPCC.
H_3	:	A set of hash functions with the $universal_2$-property.
$H(X)$	:	Entropy function.
$e(x)$	:	Function that models Eve's eavesdropping, thus giving her k bits of information.
$f(x)$	:	Function that models Eve's gain from error correction, giving her l bits of information.
$g(x)$	:	$Universal_2$ hash function in privacy amplification.
$h(p)$	:	Binary entropy function.
$f(A)$	:	A function randomly mapping A to smaller blocks.
$I(A; E)$	:	Eve's mutual information about A given from E.
$I(B; E)$	:	Eve's mutual information about B given from E.
$I(A; B)$	:	Bob's mutual information about A given from B.