QUANTUM KEY DISTRIBUTION PROTOCOLS

by

Utkan Güngördü

A Thesis Submitted to the
Graduate School of Sciences
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Physics

Koç University

August, 2010

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Utkan Güngördü

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Prof. Tekin Dereli

_____

Assoc. Prof. Özgür Müstecaplıoğlu

_____

Dr. Muhammet Ali Can

Date: _____

ABSTRACT

A review of quantum key distribution protocols has been made. A simulation of BB84 protocol with reconciliation and privacy amplification has been presented. Intervension of an eavesdropper has been implemented.

ÖZETÇE

Kuantum anahtar dağıtım protokollerinin incelemesi yapıldı. Uzlaşma ve güvenlik yükseltimi aşamalarını da içeren, BB84 protokolünün benzetimi sunuldu. İletişime yabancı müdahale gerçeklendi.

ACKNOWLEDGMENTS

The author wishes to express his thanks to Prof. Tekin Dereli for his support and advices during his education in Koç University.

TABLE OF CONTENTS

LIST OF TABLES

NOMENCLATURE

QKD	Quantum Key Distribution
QC	Quantum Cryptography
QBER	Quantum Bit Error Rate
BB84	Bennett and Brassard's 1984 protocol
B92	Brassard's 2-state 1992 protocol
E91	Ekert's 1991 EPR protocol
PNS	Photon Number Splitting attack
log	Logarithm in base 2
Alice	Transmitting party
Bob	Recieving party
Eve	Eavesdropper

Chapter 1

INTRODUCTION

1.1 An Overview of Classical Cryptography

Cryptography is the study of writing messages in such a way that its contents can only be understood by the intended receiver. The security of these methods usually relies on limited computational resource or time of eavesdroppers.

There is one classical algorithm, called one-time pad, whose security can be proved unconditionally. When this method is used, an eavesdropper can gain no information about the message beyond it's length. Suppose Alice —the sending party— has a message she would like to deliver in secrecy, converted to a binary representation. Alice prepares a binary random key, that has the same length as the message, and performs a bitwise XOR with the message. Bob —the receiving party— with the knowledge of the secret key, can decode the message by XORing the encoded text with the key again. Without the key, decoding a bit is like a coin-toss. Since every bit is encoded independently, the chances for accurate decoding decreases exponentially with the message length, so the algorithm is considered to be secure [1]. The problem with one-time pad is, the length of the required key is as long as the message itself, which is a problem for practical uses. That aside, Alice should be able to send the secret key to Bob. This of course raises the question "If she has a way of sending the key in secrecy, why doesn't she use it for transmitting the message itself, since they are of the same length anyway?". One-time pad is not meant for actual use, nevertheless, it serves as an example of unconditionally secure classical encryption. [2]

The distribution of a key in secrecy is a serious problem in practice, and has lead to a

[1] Which is a part of the unconditional security definition.

[2] It is however possible to perform this scheme in quantum cryptography —quantum key distribution solves exactly this problem.

class of ciphers that uses different keys for encryption and decryption, called *asymmetric cryptography*. The key used for encryption is called *public key* (which is to be distributed to everyone), and the key for decryption is called *private key* (which is never meant for distribution). These two keys are related to each other, but they are designed in such a way that deriving the private key from public key is computationally very difficult, relying on a one-way function.

Classical algorithms used in public-key encryption rely on computational difficulty of certain problems, such as integer factorization and discrete logarithm problems. Today, there is no algorithm that can solve these problems in polynomial time, however it is not proved that any such algorithm cannot be found either. A breakthrough can render all widely used ciphers overnight!

One other problem is the requirement for random numbers in cryptography. Good random number generation, the problem of finding uncorrelated sequence of random numbers has proved to be notoriously difficult classically.

1.2 Origins of Quantum Cryptography

Quantum cryptography is an umbrella term for applications of quantum physics to cryptography related subjects, such as digital signature, fingerprinting, random number generation, secret key distribution [3], bit commitment, oblivious transfer,... It was initiated by Stephen Wiesner in 1970 [1] in a paper in which he described the basic ideas for quantum key distribution, but the paper was not published until 1983. [4]

In 1984, Charles H. Bennett and Gilles Brassard picked up the idea and described a protocol for quantum key distribution, now known as BB84 protocol [2]. Alice prepares a random bit, and prepares a photon either in rectilinear or diagonal basis, again picked at random. She then transmits the photon to Bob, who will measure the photon in rectilinear or diagonal basis randomly. After all the photons are sent and measurements are complete, they disclose their bases used in preparation (Alice) and measurement (Bob), but not the original bits or measurement results. If Bob measured a photon in the "wrong" basis they discard that bit, since the measurement result would be probabilistic. After they perform

[3] Generation, actually.

[4] It was rejected by IEEE Information Theory, and published by SIGACT News later on.

information reconciliation and privacy amplification on the remaining bits for reasons which we will discuss later on, both parties will have a secure key on which they agree. Although the protocol was originally described using polarization states of photon, any pair(s) of conjugate states can be used in principle.

The idea was initially seen mostly as work of a science-fiction, including Bennett and Brassard, because the required technology was out of reach [5]. Initial designs such as quantum bank notes required long-term quantum memories, that is, the technology to store a photon for days without significant decoherence. It was later on realized that fully working systems can be designed with the idea that photons should only be used to *transmit* data, not to *store* it.

The security of this protocol lies on the fact that, if eavesdropper interferes with the photon she would disturb it's state, eventually revealing her presence (for instance Bob can measure the incoming photon in the "correct" basis, but because of Eve's intervention, may find a different result). This means that the security is based on the laws of quantum mechanics, and not computational infeasibility. Considering the fact that classical cryptosystems are based on a mixture of guesswork and mathematics, this is an important milestone.

In 1991, initially unaware of the earlier work, Arthur Ekert invented a new way for quantum key distribution independently. The protocol exploits the non-local nature of quantum mechanics, first described by Einstein, Podolsky and Rosen in a 1935 paper [7]. This protocol requires a source that emits particles in a maximally entangled state

$$|\Psi^-\rangle = \frac{|\uparrow\downarrow\rangle - |\uparrow\downarrow\rangle}{\sqrt{2}}. \tag{1.1}$$

Alice and Bob each gets one particle from each pair. Like Bob does in BB84 protocol, Alice and Bob measure the incoming particles in a random state, and afterwards, they announce their bases used in measurement. For measurements with agreeing bases, they should expect opposite results, due to the nature of the state they use. After either Alice or Bob inverts her/his agreeing bits, they should have identical keys.

Remaining measurement results, where they used different bases for measurement, can be used to detect the presence of an eavesdropper. Each party, using the measurement results of non-agreeing bases, can check whether if Bell inequality (see 2.4.3) holds or not.

If the inequality is not violated, it would mean the particles were not perfectly entangled, implying the disturbance of an eavesdropper.

1.3 Why Do We Need QKD?

Classical public-key (asymmetrical) encryption schemes rely on NP problems, which cannot be solved efficiently on a classical computer, such as integer factorization and discrete logarithm problem. These problems share the property that once we have a candidate for solution, it can be checked efficiently, but coming up with an exact solution cannot be achieved in polynomial time. It has been found, however, that this is not the case for a quantum computer. Peter Shor found that both integer factorization and discrete logarithm problems can be solved efficiently on a quantum computer [15], jeopardizing the security [5] of widely-used public-key encryption schemes like RSA. And this is why wide-spread usage of RSA is alerting, demanding a better solution for secure communication over public channels, before devices implementing Shor's factoring algorithm appear in the market.

One-time pad is an absolutely secure cryposystem, and block ciphers such as AES do not exploit the mentioned NP problems and are considered to be safe up-to-day. But these algorithms use the same key for both encryption and decryption, which means Alice will have to find a way to send her key to Bob in a secure way. Quantum key distribution addresses this issue, providing a method for secret key generation between two parties. Moreover, as it turns out, QKD over a public channel can be *proved* to be secure, unconditionally. This security is guaranteed by the laws of quantum mechanics.

1.4 Structure of This Thesis

This thesis is divided into 7 chapters. In chapter one, we give a brief background history as a motivation. In chapter 2, we review the preliminaries required for quantum key distribution, namely the classical information theory and basic quantum mechanics for QKD. We introduce widely-known QKD protocols in chapter 3, followed by information reconciliation in chapter 4 and privacy amplification in chapter 5. For completeness, we briefly discuss

[5] Here, we mean conditional security, such as: given that integer factorization is NP and eavesdropper has limited computational power, RSA is secure.

some of the security related issues in chapter 6. Finally, in chapter 7, we give the results of our simulation, whose code is presented in appendix.

Chapter 2

PRELIMINARIES

2.1 Classical Information Theory

In information theory, the stream of data is modelled as a sequence of independent random numbers from a certain probability distribution. And Shannon entropy —which will be a particular function of interest— for a discrete random number sequence is defined by

$$H(X) \equiv -\sum_i p(x_i) \log p(x_i), \tag{2.1}$$

[1] where $p(x_i)$ is the probability for appearance of symbol x_i, and $X = \{x_i\} = \{x_1, x_2, \ldots, x_n\}$ is a vector of symbols. This function can be seen as the average of $\log(x_i)$ over the probability distribution $\{p(x_i)\}$.

One of the essential ideas in the definition above is this. If a symbol is less likely to occur, it carries more information or has a higher "surprise factor", and the associated term should contribute more. A series of 0s would contain the least possible information for instance ($p = 1$ and $H = 0$). One may be inclined to think that the sequence 01010101... shouldn't contain any information at all either by this rationale, since it's a series of 01s (or 010101s), but it is clearly not the case if we consider single bits as our symbol table (in which case we get $H = 1$). Shannon entropy depends on the choice of symbol table (or "dictionary"), and is of important consideration when one tries to compress data, as in the case of Lempel-Ziv family algorithms, but since we will be restricting ourselves to binary alphabet in QKD, we will not be discussing them.

One other important property of entropy function is related to additivity of information. When two independent events, p and q, occur together, one would expect to gain information $I(p) + I(q)$, where $I(p) \equiv \log(p)$ is the information gained when event p occurs individually. In other words,

[1] Unless explicitly stated, $\log(x)$ refers to $\log_2(x)$.

$$I(pq) = I(p) + I(q), \tag{2.2}$$

and it is all thanks to additivity property of logarithm function. Moreover, this establishes an intuitive interpretation for Shannon entropy: it is the average information gain for independent events with occurrence probabilities $\{p(x_i)\}$.

One other interpretation comes from Shannon's *source coding theorem*, which we will state here without giving proof.

Theorem 2.1.1. (Shannon's source coding theorem) *As the data length grows large, lowest possible code rate (average bits per symbol) that can be achieved by a lossless compression is the Shannon entropy of the source.*

This means that on average, compressed size of a message will be $|X|H(X)$ bits, where $|X|$ is the length [2] of the message.

One trivial case for Shannon entropy is where every possible symbol has the same chance of occurrence. In this case, each term contributes equally with probability $p(x_i) = 1/|X|$, resulting in

$$H(X) = \log |X|. \tag{2.3}$$

For any other distribution, this becomes an inequality

$$H(X) \leq \log |X|. \tag{2.4}$$

One of the common distributions is the binary distribution, and we will define a special function for its entropy

$$h(p) = -p \log p - (1-p) \log(1-p) \tag{2.5}$$

where p is the probability of occurrence for one of the symbols.

Other entropies can be defined in the case of two random variables.[3] We can define *joint*

[2] Number of symbols.

[3] These definitions can be extended to more than two variables.

entropy using the joint probability distribution of X and Y, $p(x_i, y_j)$

$$H(X,Y) \equiv -\sum_{i,j} p(x_i, y_j) \log p(x_i, y_j) \tag{2.6}$$

If X and Y are not correlated, their joint probability distribution can be written as the product of marginal distributions $p(x_i, y_j) = p_X(x_i)p_Y(y_j)$. It can be shown that joint entropy satisfies the inequality

$$H(X,Y) \leq H(X) + H(Y), \tag{2.7}$$

with equality if and only if X and Y are independent random variables. In terms of source coding theorem, this means that two sources can be compressed better jointly, rather than compressing each one individually.

When we know the value Y, our "surprise factor" should drop by $H(Y)$ compared to the case when we don't know X or Y. Remaining Shannon entropy is called the *conditional entropy*, defined as

$$H(X|Y) \equiv H(X,Y) - H(Y) \tag{2.8}$$

$$H(X|Y) = -\sum_{i,j} p(y_j)p(x_i|y_j) \log p(x_i|y_j) = H(X,Y) - H(Y) \tag{2.9}$$

And the mutual information is defined as

$$H(X:Y) \equiv H(X) + H(Y) - H(X,Y) \tag{2.10}$$

As an example, we may consider a noisy channel on which we can transmit 0s or 1s. We shall assume that physical properties of the channel are symmetric, so that the "bit-flip" probability (or crossover probability) is the same no matter which "signal" is sent (such a channel is called *binary symmetric channel*). If the error probability is ϵ, then

$$p(y_j = 0|x_i = 1) = p(y_j = 1|x_i = 0) = \epsilon \tag{2.11}$$

$$p(y_j = 0|x_i = 0) = p(y_j = 0|x_i = 0) = 1 - \epsilon \tag{2.12}$$

Since $p(x, y) = p(x)p(y|x)$ and $H(Y) = 1$, mutual information of the channel is found to be

$$H(X : Y) = H(Y) - \sum_{i,j} p(x_i)p(y_j|x_i) \log p(y_j|x_i) = 1 - h(\epsilon). \tag{2.13}$$

In quantum key distribution protocols, we will assume that the error sources (which is typically the eavesdropper, the "wrong-basis errors due to Bob" are eliminated in the sifting stage) are symmetric in the sense that the way photon states are disturbed does not depend on the initial state of the photon. When we analyze the security of the BB84 protocol, we will refer back to this result.

2.1.1 Shannon's Noisy-Channel Coding Theorem

When we try to communicate using a noisy channel, we usually want the recipient to get a certain message without any deformations. There are many schemes to correct errors in a message, which requires us to add some extra bits to the message for error correction purposes. We usually want to keep the number of this excess bits at minimum, because error correction data itself can also be corrupted and if its length gets larger, the number of errors increases. And we would like to keep message efficiency as good as possible, that is we would like to reduce the number of excess bits per message bit in a transmission.

More formally, let's say the message we would like to send is M bits, and the number of extra bits (code bits) we require for a certain error correction scheme is m. Then the total message length for transmission is $M_C = M + m$. We would like to keep the ratio $C = M/M_C$ (the *transmission rate*) as big as possible, but at the same time be able to communicate with arbitrarily small number of uncorrected errors.

In 1948, Claude Shannon showed that there is a limit to how small M/M_C can be made, independent of the error correction scheme. He showed that the following inequality holds for this ratio:

$$M/M_c \leq 1 - h(\epsilon) \tag{2.14}$$

where ϵ is the probability for a single bit going wrong. This is a very important theorem, defining the limits of communication over all kinds of channels with noise.

We adopt a non-rigorous proof following the one given in [18]. We assume that the total message M_C is large, so that expected values converge to actual numbers (or otherwise, we can imagine that we're repeating the same communication scheme many times over and consider the average of consecutive trials). Since the error probability for one bit is ϵ, the (average) number of error in the whole message will be

$$k = \epsilon M_C \tag{2.15}$$

The number of ways this k errors could be distributed though the whole message is given by

$$\frac{M_C!}{k!(M_C - k)!} \tag{2.16}$$

And an error in a bit is nothing more than a flip, we only need to know its location to correct it. The information for the locations of these errors should be conveyed within m code bits, which can mean 2^m different things, and it should at least contain the exact locations of possible distributions of errors, which means

$$2^{M_C - M} \geq \frac{M_C!}{k!(M_C - k)!} \tag{2.17}$$

Taking the logarithm and using Stirling's approximation, it results in

$$M_C - M \geq M_C \left(-\epsilon \log \epsilon - (1 - \epsilon) \log(1 - \epsilon)\right) \tag{2.18}$$

which is equivalent to the original statement. Comparing with the previous result (2.13), we see that mutual information is the limiting value for transmission rate, that is average bits per channel use. To illustrate Shannon's limit, we list the coding efficiency or transmission rate M/M_C for various values of channel error ratio ϵ:

For the extreme case of $\epsilon = 1/2$, the noise dominates over the useful content, making reliable transmission impossible. A possible error rate for wireless modems at 40-45 meters is 1/1000, which means we need to add about at least 12 code bits to send 1000 bits of data.

ϵ	M/M_C	$(M_C - M)/M$
1/2	0	∞
1/3	0.082	11.2
1/10	0.531	0.9
1/100	0.919	0.09
1/1000	0.988	0.012

Table 2.1: Message efficiency and code bit ratio for various values of error rate.

2.2 Bits in Quantum Mechanics

Classically, information is represented by numbers, usually written in binary representation. A *bi*nary digi*t* is called *bit*, and is the basic unit of information. An "unbiased" binary variable has a unit Shannon entropy, or contains one bit of information.[4] In communication, a bit may represent conductance state of a transistor, or a certain voltage threshold for a signal.

In quantum mechanics, a binary system is represented by a two-level system, and its state is what describes a *qubit*, short for quantum bit. A two-level system "lives" as a vector in a two-dimensional Hilbert space, whose basis vectors can be labeled $\{|0\rangle, |1\rangle\}$. Written with these labels, this basis is usually referred as the *computational basis*, or *rectilinear basis*. A general qubit in this basis can be written as

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle. \tag{2.19}$$

Throughout the text we will be using Dirac notation, a complete description of bras and kets can be found in quantum mechanics books such as [14]. Assuming the following basis

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \equiv |0\rangle, \qquad \begin{pmatrix} 0 \\ 1 \end{pmatrix} \equiv |1\rangle, \tag{2.20}$$

we can also write in vector form

[4]Since $h(1/2) = 1$. Note that if the probabilities of 0 and 1 are not equal, the information content is less than 1 bit. It is this fact which makes data compression possible.

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{2.21}$$

Here, α and β are complex numbers, satisfying the normalization condition $\||\psi\rangle\|^2 = \langle\psi|\psi\rangle = |\alpha|^2 + |\beta|^2 = 1$, and their physical meaning is as follows. If we carry out a measurement in $\{|0\rangle, |1\rangle\}$ basis, the outcome will be $|0\rangle$ with probability $|\alpha|^2$, and $|1\rangle$ with probability $|\beta|^2$. They are usually called *probability amplitudes* in physical contexts, referring to the interference of these "amplitudes" when there are more than one particles. Using the normalization condition, this state can also be written as

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle \tag{2.22}$$

Since qubits can be represented by two dimensional vectors, any operation on them can be represented by 2×2 unitary matrix (unitarity is essential because of the normalization condition). Any 2×2 matrix can be represented as linear combination of Pauli matrices

$$\sigma_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \sigma_2 = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.23}$$

and a 2×2 unit matrix

$$\sigma_0 = I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{2.24}$$

Pauli matrices were originally defined to describe the spin state of an electron, which is a two-level system. These matrices are generators of $SU(2)$ rotations (up to a constant, i). They obey the product rule

$$\sigma_j\sigma_k = \delta_{jk} + i\sum_{l=1}^{3}\epsilon_{jkl}\sigma_l \tag{2.25}$$

which implies, along with the fact that Pauli matrices are traceless,

$$\mathrm{tr}(\sigma_j\sigma_k) = 2\delta_{jk}. \tag{2.26}$$

Using these, we can break any 2×2 matrix into it's "Pauli components" as follows. Since

M can be written in terms of Pauli matrices and unit matrix, we can write

$$M = M_0\sigma_0 + M_1\sigma_1 + M_2\sigma_2 + M_3\sigma_3 \tag{2.27}$$

Multiplying this equation from left with σ_j, and taking the trace, we have

$$\mathrm{tr}(\sigma_j M) = \mathrm{tr}\left(\sigma_j \sum_{k=0}^{3} M_k\sigma_k\right) = 2M_j \tag{2.28}$$

A photon's polarization state is a two-level system, and represents a qubit. All operations on its polarization state can be described in terms of Pauli matrices and a unit matrix.

The $\{|0\rangle, |1\rangle\}$ basis is usually taken to be eigenstates of σ_3 operator with eigenvalues $+1$ and -1 respectively. One other important basis used in QKD is the eigenstates of σ_1 operator,

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{2.29}$$

$$|-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \tag{2.30}$$

again with eigenvalues $+1$ and -1 respectively. As expected, the basis states are orthogonal, meaning $\langle +|-\rangle = 0$. An important fact exploited by BB84 is the fact that these states are not orthogonal to computational basis vectors, and a measurement in computational basis cannot distinguish σ_1 eigenstates at all:

$$|\langle +|0\rangle|^2 = |\langle +|1\rangle|^2 = |\langle -|0\rangle|^2 = |\langle -|1\rangle|^2 = \frac{1}{2} \tag{2.31}$$

which behaves like a fair coin toss, having the highest average "surprise factor" (i.e., Shannon entropy).

σ_1 is also called the *bit-flip operator*, due to its effect on computational basis vectors

$$\sigma_1|0\rangle = |1\rangle \tag{2.32}$$

$$\sigma_1|1\rangle = |0\rangle \tag{2.33}$$

Bit-flip errors in a quantum channel can be modeled using this operator (and the unit operator). σ_3 does the analogous thing for $|+\rangle$ and $|-\rangle$ vectors.

$\sigma_1, \sigma_2, \sigma_3$ matrices are sometimes written as $\sigma_x, \sigma_y, \sigma_z$ (after electron's spin components). For this reason $\{|0\rangle, |1\rangle\}$ is also referred as the Z basis, and $\{|+\rangle, |-\rangle\}$ as the X basis. And their eigenstates with eigenvalue ± 1 is denoted as $|\pm z\rangle$ and $|\pm x\rangle$. We will use both notations throughout the text.

2.2.1 Bloch Sphere Representation of a Qubit

It can be easily verified by direct substitution that the two-angle representation of a qubit

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle \tag{2.34}$$

is actually the eigenstate of the operator $\boldsymbol{\sigma}\cdot\hat{\boldsymbol{n}}$ with eigenvalue $+1$, where

$$\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \sigma_3) \tag{2.35}$$

$$\hat{\boldsymbol{n}} = (u, v, w) = (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta) \tag{2.36}$$

This unit vector $\hat{\boldsymbol{n}}$ is called the *Bloch vector*, which spans a unit sphere with angular coordinates (θ, ϕ). It is a representation of a qubit as a point on a unit sphere in a fictitious three-dimensional space.

2.2.2 Coherent States

It is mathematically straightforward to encode a qubit using its polarization state, since it readily is a two-level system. For realization of BB84 any 2-level system can be used in theory. In practice, almost all implementations encode qubits using photons. Thanks to the developments in optical telecommunication, decoherence of photons can be controlled and moderated.

However, producing single photons is not an easy task. In most practical applications, sending party encodes her random bits using weak laser pulses, a feasible way of producing single photons with fair probability. Generated pulses can be described by coherent states [19]. We will not discuss the physics of weak laser pulses, and how they eventually output

photons in coherent states, as the topic is outside of the scope of this thesis. We will, however, mention some properties of coherent states briefly.

Electromagnetic field can be described as an assembly of independent harmonic oscillators [5], a full treatment on quantization of electromagnetic field can be found in quantum mechanics text books, such as [17]. We only quote the relations that we will need later on.

A coherent state $|\lambda\rangle$ is the eigenstate of the annihilation operator a, satisfying the eigenvalue equation

$$a|\lambda\rangle = \lambda|\lambda\rangle \tag{2.37}$$

and it's explicit solution is

$$|\lambda\rangle = e^{-|\lambda|^2/2} \sum_{n=0}^{\infty} \frac{\lambda^n}{\sqrt{n!}} |n\rangle \tag{2.38}$$

where the state $|n\rangle$ represents the case where "there are n photons in the electromagnetic field" [6], $\mu = |\lambda|^2$ is the average number of photons $\langle n\rangle = \langle\lambda|a^\dagger a|\lambda\rangle$, or the *intensity*. Clearly, the probability of finding n photons in the field is

$$P(n) = e^{-\mu} \frac{\mu^n}{n!} \tag{2.39}$$

which happens to be the Poissonian probability distribution.

To get close to the single photon limit, we may like to make μ very small, but then we hardly get any photon at all, since for $\mu \ll 1$, $P(0) \approx 1 - \mu$ in this limit. For the rare cases where we get *some* photons, the chances of getting only one photon is $P(n = 1|n > 0) = P(1)/(1 - P(0)) = e^{-\mu}\mu/(1 - e^{-\mu}) \approx 1 - \mu/2$. In a practical demonstration, attenuation reduced intensity down to $\mu \approx 0.12$ [5], which means for about 88% of the cases we get no photon at all. Within the remaining non-empty pulses, we only get about 6% single photon states.

Although coherent states can be easily created, the higher order terms in the state can pose serious problems. We see that there's a non-zero chance in a weak laser pulse to have

[5] Harmonics oscillator and particle approaches are equivalent, see for instance *Feynman Lectures on Physics Vol. III, 4-5.*

[6] They are also referred as *Fock states*, after Soviet physicist V.A. Fock.

more than one photons, produced in the same state. This is a security hole, because an eavesdropper can save one of the extra photon(s) in a quantum memory and send the rest to Bob, wait until the bases are announced, and perform the correct measurements. This attack, called *photon number splitting attack* (PNS), allows Eve to gain information without being detected at all. This means, in weak laser pulse implementations, one must take into account the existence of multiphoton pulses when computing Eve's information, and QBER threshold.

2.3 Quantum Information Theory

In this section, we briefly introduce some basic tools of quantum information theory that we will need later on.

2.3.1 Density Operator

Instead of using state vectors in Hilbert space, it is possible to formulate quantum mechanics in terms of what is called *density operator* (or *density matrix*). It is particularly useful when considering a quantum system's interaction with its environment.

Let us assume that a quantum system state would be prepared in $|\psi_i\rangle$ with probability p_i (where is goes from 1 to N). Then the density operator ρ of this system is defined as

$$\rho \equiv \sum_{i=1}^{N} p_i |\psi_i\rangle\langle\psi_i|. \tag{2.40}$$

ρ is a positive semi-definite matrix, so it is in general true that it has a spectral decomposition in terms of orthogonal unit vectors $|e_i\rangle$, which we can *always* write in the following form

$$\rho = \sum_{i} \lambda_i |e_i\rangle\langle e_i|. \tag{2.41}$$

We can express the evolution of the system and measurements on it in this formalism as well. The evolution will be described by a unitary operator U, and basis states will accordingly become $U|\psi_i\rangle$, and $\bar{\rho}$ becomes

$$\rho' = \sum_i p_i U|\psi_i\rangle\langle\psi_i|U^\dagger = U\rho U^\dagger \tag{2.42}$$

$$p(m) = \sum_i p(m|i)p_i \tag{2.43}$$

Since given the initial state $|\psi_i\rangle$ the probability of obtaining m is $\langle\psi_i|M^\dagger M|\psi_i\rangle$,

$$p(m) = \sum_i p_i\langle\psi_i|M^\dagger M|\psi_i\rangle \tag{2.44}$$

$$= \sum_{i,j} p_i\langle\psi_i|j\rangle\langle j|M^\dagger M|\psi_i\rangle \tag{2.45}$$

$$= \mathrm{tr}\left(M^\dagger M\rho\right) \tag{2.46}$$

Measurement operators act in the same form as unitary operators, except that they do not preserve the unitarity, so we can write down the post-measurement density operator, up to a constant C, this way

$$\bar{\rho} = CM\rho M^\dagger \tag{2.47}$$

We can work out the constant like this. The unitarity condition, in terms of density operator, is

$$\mathrm{tr}(\rho) = \sum_i p_i \mathrm{tr}\left(|\psi_i\rangle\langle\psi_i|\right) = \sum_i p_i = 1 \tag{2.48}$$

and since the post-measurement density matrix should obey this condition as well,

$$\mathrm{tr}\left(\bar{\rho}\right) = \mathrm{tr}\left(CM\rho M^\dagger\right) = 1 \tag{2.49}$$

which means the constant C is $1/\mathrm{tr}\left(M^\dagger M\rho\right)$, resulting in

$$\bar{\rho} = \frac{M\rho M^\dagger}{\mathrm{tr}\left(M^\dagger M\rho\right)}. \tag{2.50}$$

The expectation value of an operator A is

$$\langle A\rangle = \sum_i p_i \langle\psi_i|A|\psi_i\rangle = \mathrm{tr}\left(\rho A\right). \tag{2.51}$$

A density operator satisfies

$$\rho^2 = \rho \qquad \text{and} \qquad \mathrm{tr}\left(\rho^2\right) = 1 \tag{2.52}$$

if and only if it is a *pure state*. An example would be $\rho = |\psi\rangle\langle\psi|$. Otherwise, it is a composition of pure states (such as $\rho = p|\psi\rangle\langle\psi| + (1-p)|\phi\rangle\langle\phi|$), and is called a *mixed state*, and satisfies

$$\mathrm{tr}\left(\rho^2\right) < 1 \tag{2.53}$$

Since we will be working with qubits, it is of interest to know the density matrix of a qubit. Written in terms of "Bloch angles" (2.34) , we can right away tell the result of $\sigma_i|\psi\rangle$, since it is an eigenstate with "components" given in the corresponding Bloch vector. Using (2.28) with this fact, we have

$$\rho = |\psi\rangle\langle\psi| = \frac{1}{2}\left(I + \boldsymbol{\sigma}\cdot\boldsymbol{r}\right) \tag{2.54}$$

2.3.2 Fidelity

One of the most commonly used way of measuring the "closeness" of two given quantum states is *fidelity*. The fidelity between two quantum states ρ and σ is defined to be

$$F(\rho,\sigma) \equiv \mathrm{tr}\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}} \tag{2.55}$$

and lies within the interval

$$0 \le F(\rho,\sigma) \le 1 \tag{2.56}$$

There are some remarkable cases we would like to mention When $\rho = \sigma$, we have $F = 1$. For $\rho = |\psi\rangle\langle\psi|$, a pure state, we have

$$F(\rho,\sigma) = \mathrm{tr}\sqrt{\langle\psi|\sigma|\psi\rangle|\psi\rangle\langle\psi|} = \sqrt{\langle\psi|\sigma|\psi\rangle} \tag{2.57}$$

As a special case is when σ is pure state as well (say, $|\phi\rangle\langle\phi|$), which yields $F(\rho,\sigma) = |\langle\psi|\phi\rangle|$. When two states commute, that is $[\rho,\sigma] = 0$, both density matrices can be diagonalized together

$$\rho = \sum_i r_i |e_i\rangle\langle e_i|, \qquad \sigma = \sum_i s_i |e_i\rangle\langle e_i| \tag{2.58}$$

and fidelity becomes

$$F(\rho,\sigma) = \mathrm{tr}\sqrt{\sum_i r_i s_i |e_i\rangle\langle e_i|} = \sum_i \sqrt{r_i s_i}. \tag{2.59}$$

This is the classical definition of fidelity, in terms of probability distributions $\{r_i\}$ and $\{s_i\}$.

2.4 Some Important Results of Quantum Mechanics

2.4.1 No-cloning Theorem

No-cloning theorem is the backbone of the security of BB84 protocol, which forbids making identical copies of an arbitrary, unknown state. We will prove the theorem on mathematical grounds, but prior to that, we would like to mention an informal proof based on Heisenberg's uncertainty principle. Suppose we are able to make identical copies of a particle's state. Then we could measure the position of the original particle with arbitrary precision, and perform a momentum measurement on the clone, thus violating the uncertainty principle. We see that security of BB84 is protected by the uncertainty principle. The actual proof goes as follows.

Assume that we have a "cloning machine", that can copy the *unknown* state of a particle ($|\psi\rangle$) onto another, which we can refer as "blank qubit" ($|0\rangle$). We assume that the machine works independent of the original state. This operation can be denoted as

$$|\psi\rangle|0\rangle \to |\psi\rangle|\psi\rangle \tag{2.60}$$

Like any other operation in quantum mechanics, we should be able to represent this as a unitary operation [7]

[7] This stems from the fact that probabilities should add up to 1.

$$U|\psi\rangle|0\rangle = |\psi\rangle|\psi\rangle \tag{2.61}$$

Since this is a universal cloning machine, the same relation should hold for another —again, initially unknown— state $|\phi\rangle$

$$U|\phi\rangle|0\rangle = |\phi\rangle|\phi\rangle \tag{2.62}$$

When we take the inner product of both sides of these equations, we get

$$\langle 0|\langle\psi|U^\dagger U|\phi\rangle|0\rangle = \langle\psi|\langle\psi|\phi\rangle|\phi\rangle \tag{2.63}$$

For a unitary operator, $U^\dagger U$ is unit operation I by definition, so this equation is actually

$$|\langle\psi|\phi\rangle|^2 = \langle\psi|\phi\rangle \tag{2.64}$$

In other words, either this machine works for one kind of state ($\psi = \phi$), or $\langle\psi|\phi\rangle = 0$. They both imply we already know the input state in advance, and work only for particular states, which means this is not a universal cloning machine.

Theorem 2.4.1. (No-cloning theorem) *It is impossible to create identical copies of an unknown, arbitrary quantum state.*

There is, however, work done on creating imperfect copies of quantum states, which are usually based on minimizing the fidelity (see 2.3.2) between the original state and clone. See for instance [20, 21]. These so-called cloning machines can be used for designing attacks in BB84 [21], but we won't be discussing them in this thesis.

2.4.2 Information Gain Implies Disturbance

No-cloning theorem is not the only physical constraint on eavesdropping, however. One can also show that in Eve's attempts to distinguish between the non-orthogonal states, her gaining information is only possible at the expense of disturbing the qubit. We can show that this proposition is not limited to bases used in BB84, but true for *any* non-orthogonal pair. Let us say $|\psi\rangle$ and $|\phi\rangle$ are such pair. Without loss of generality, we can assume that

she uses an ancillary qubit to obtain information on the incoming qubit [15]. If we label the initial state of the ancilla $|u\rangle$, and her "method" for obtaining information U, a unitary operator which does not disturb the incoming state at all , we can write

$$U|\psi\rangle|u\rangle = |\psi\rangle|v\rangle \tag{2.65}$$

$$U|\phi\rangle|u\rangle = |\phi\rangle|v'\rangle \tag{2.66}$$

For her to obtain *some* information about the incoming state, $|v\rangle$ and $|v'\rangle$ should differ, so that she can use this difference to acquire information about the incoming unknown (to her) state. Taking the inner products side-by-side, and using the fact that $U^\dagger U = I$, we have

$$\langle u|u\rangle\langle\psi|\phi\rangle = \langle v|v'\rangle\langle\psi|\phi\rangle \tag{2.67}$$

Also, by taking the inner products to the equations by themselves, we must —due to normalization— have

$$\langle u|u\rangle = 1 \tag{2.68}$$

$$\langle v|v'\rangle = 1. \tag{2.69}$$

Which means $|v\rangle = |v'\rangle$. In other words, it is impossible for Eve to gain some sort of information about the incoming state without causing any disturbance in the incoming qubit.

It is this fact which makes it possible to construct a quantum key distribution protocol that uses only two non-orthogonal states. Such a protocol (B92) is described in detail in 3.3.

2.4.3 Bell Inequality

In 1935, Einstein, Podolsky and Rosen have written a paper in which they concluded that the physical reality as described by quantum mechanics is incomplete [7]. They considered a pair of entangled particles (an *entangled state* is a multipartite state which cannot be broken

into product of single particle states), which interacted during a certain interval of time, and stopped interacting afterwards. After waiting long enough time, an observer measures one of the particles, causing the wavefunction collapse. Following quantum mechanics, we instantaneously make the other particle to be in a certain state as well. For this reason, Einstein referred entanglement as "spooky action at-distance".

We can, for instance, consider a pair of an electron and a positron in the spin state

$$|\Psi^-\rangle = \frac{|0\rangle|1\rangle - |1\rangle|0\rangle}{\sqrt{2}}. \tag{2.70}$$

[8] Suppose we measured the spin of the first particle in Z basis, and found $|1\rangle$. Then we can tell that the second particle will be in the $|0\rangle$ state right away, and this happens instantaneously, no matter what the spatial separation between the particles may be.

The idea was, that, both particles had some extra information from the beginning, and this was how the observer could tell about the second particle instantaneously. It's just that wavefunction does not contain this extra bit of information (or hidden variables, λ), because quantum mechanics is incomplete.

At the time, it was thought that the predictions of quantum mechanics is correct, but the results are due to the statistical distribution of hidden variables —something missing in quantum mechanics.

In 1964, John S. Bell showed that *any* (local) theory of hidden variables is incompatible with quantum mechanics [8]. That is to say, one can either conclude that quantum mechanics is downright wrong or there are no local hidden variables —"quantum mechanics is incomplete" is not an option here. We will closely follow [8] in demonstrating this fact here.

Considering the experiment with the singlet state $|\Psi^-\rangle$, the prediction of quantum mechanics for $E(\boldsymbol{a}, \boldsymbol{b}) = \langle(\boldsymbol{\sigma}_1 \cdot \hat{\boldsymbol{a}})(\boldsymbol{\sigma}_2 \cdot \hat{\boldsymbol{b}})\rangle$ [9] (this function is also referred as *correlation function*) is

$$\langle\Psi^-|(\boldsymbol{\sigma}_1 \cdot \hat{\boldsymbol{a}})(\boldsymbol{\sigma}_2 \cdot \hat{\boldsymbol{b}})|\Psi^-\rangle = -\boldsymbol{a} \cdot \boldsymbol{b} \tag{2.71}$$

where, $\boldsymbol{\sigma}_i$ denotes the Pauli operators as components of a vector (as in 2.35) for the ith

[8] This state is sometimes called *singlet state*.

[9] We consider measurement of spin in units of $\hbar/2$.

particle. $\boldsymbol{a}$ and $\boldsymbol{b}$ are unit vectors. Bell showed that no local hidden variable theory can be compatible with this result.

We assume that A, the measurement result of $\boldsymbol{\sigma}_1 \cdot \boldsymbol{a}$, depends only on $\boldsymbol{a}$ and hidden variables represented by λ, and similarly B ($\boldsymbol{\sigma}_2 \cdot \boldsymbol{b}$ measurement) of the second particle depends on $\boldsymbol{b}$ and λ. This is a crucial step, because here we single out the possibility that the result B for particle does not depend on the measurement configuration of the first particle ($\boldsymbol{a}$), vice and versa. This is the assumption of *locality*.

Obviously, the possible values for spin measurements are

$$A(\boldsymbol{a}, \lambda) = \pm 1, \qquad B(\boldsymbol{b}, \lambda) = \pm 1. \tag{2.72}$$

Then the expectation value of $A(\boldsymbol{a}, \lambda)B(\boldsymbol{b}, \lambda)$ in this framework is

$$E(\boldsymbol{a}, \boldsymbol{b}) = \int d\lambda A(\boldsymbol{a}, \lambda)B(\boldsymbol{b}, \lambda) \tag{2.73}$$

where ρ is the probability distribution function of the hidden variable. We do not make any assumption about the nature of $\rho(\lambda)$ except that it is a normalized distribution function, that is

$$\int d\lambda \rho(\lambda) = 1. \tag{2.74}$$

Since when the detectors are aligned, that is when $\boldsymbol{a} = \boldsymbol{b}$, the measurement results will be perfectly anti-correlated, we learn the following about the form of measurement functions

$$A(\boldsymbol{a}, \lambda) = -B(\boldsymbol{a}, \lambda). \tag{2.75}$$

Using this, we can re-write (2.73):

$$E(\boldsymbol{a}, \boldsymbol{b}) = -\int d\lambda \rho(\lambda) A(\boldsymbol{a}, \lambda) A(\boldsymbol{b}, \lambda) \tag{2.76}$$

Let $\boldsymbol{c}$ be another unit vector. Then we can write

$$E(\boldsymbol{a}, \boldsymbol{b}) - E(\boldsymbol{a}, \boldsymbol{c}) = -\int d\lambda \rho(\lambda) \left[A(\boldsymbol{a}, \lambda) A(\boldsymbol{b}, \lambda) - A(\boldsymbol{a}, \lambda) A(\boldsymbol{c}, \lambda)\right] \tag{2.77}$$

$$= -\int d\lambda \rho(\lambda) A(\boldsymbol{a}, \lambda) A(\boldsymbol{b}, \lambda) \left[1 - A(\boldsymbol{b}, \lambda) A(\boldsymbol{c}, \lambda)\right] \tag{2.78}$$

Here, we used the fact that $A(\boldsymbol{b},\lambda)^2 = 1$. Since A can be either 1 or -1,

$$|E(\boldsymbol{a},\boldsymbol{b}) - E(\boldsymbol{a},\boldsymbol{c})| \leq \int d\lambda\rho(\lambda)[1 - A(\boldsymbol{b},\lambda)A(\boldsymbol{c},\lambda)] \tag{2.79}$$

or

$$1 + E(\boldsymbol{b},\boldsymbol{c}) \geq |E(\boldsymbol{a},\boldsymbol{b}) - E(\boldsymbol{a},\boldsymbol{c})| \tag{2.80}$$

This is the original Bell inequality.

We can consider that case where $\boldsymbol{a}\cdot\boldsymbol{b} = \boldsymbol{b}\cdot\boldsymbol{c} = 1/\sqrt{2}$ and $\boldsymbol{a}\cdot\boldsymbol{c} = 0$. Then Bell inequality reads

$$1 - \frac{1}{\sqrt{2}} \geq \frac{1}{\sqrt{2}} \tag{2.81}$$

which is a contradiction.

Violation of Bell inequality has been experimentally verified by Alain Aspect, Philippe Grangier and Gerard Roger in 1982 [11]. This was the experimental confirmation that nature is fundamentally nonlocal, and that the "spooky action at distance" really exists.

2.4.4 CHSH Inequality (Generalized Bell Inequality)

In a more general version of Bell inequality, due to John Clauser, Michael Horne, Abner Shimony and Richard Holt, one more term is incorporated [9]. We shall adopt the derivation in [10], however.

Assuming the previous notation, we follow (2.77)

$$E(\boldsymbol{a},\boldsymbol{b}) - E(\boldsymbol{a},\boldsymbol{b}') = \int d\lambda\rho(\lambda)[A(\boldsymbol{a},\lambda)B(\boldsymbol{b},\lambda) - A(\boldsymbol{a},\lambda)B(\boldsymbol{b}',\lambda)] \tag{2.82}$$

But this time, we introduce 2 extra terms that add up to zero on the right-hand side

$$\begin{aligned} E(\boldsymbol{a},\boldsymbol{b}) - E(\boldsymbol{a},\boldsymbol{b}') = &\int d\lambda\rho(\lambda)A(\boldsymbol{a},\lambda)B(\boldsymbol{b},\lambda)[1 \pm A(\boldsymbol{a}',\lambda)B(\boldsymbol{b}',\lambda)] \\ &- \int d\lambda\rho(\lambda)A(\boldsymbol{a},\lambda)B(\boldsymbol{b}',\lambda)[1 \pm A(\boldsymbol{a}',\lambda)B(\boldsymbol{b}',\lambda)] \end{aligned}$$

Applying the triangle inequality and taking (2.72) into account, we have

$$|E(\boldsymbol{a},\boldsymbol{b})-E(\boldsymbol{a},\boldsymbol{b}')| \leq \int d\lambda\rho(\lambda)[1\pm A(\boldsymbol{a}',\lambda)B(\boldsymbol{b}',\lambda)]+\int d\lambda\rho(\lambda)[1\pm A(\boldsymbol{a}',\lambda)B(\boldsymbol{b},\lambda)] \quad (2.83)$$

Since $\rho(\lambda)$ is normalized (2.74), we have

$$|E(\boldsymbol{a},\boldsymbol{b}) - E(\boldsymbol{a},\boldsymbol{b}')| \leq 2 \pm [E(\boldsymbol{a},\boldsymbol{b}') + E(\boldsymbol{a}',\boldsymbol{b}')] \quad (2.84)$$

which includes the *CHSH inequality* (or sometimes referred as *generalized Bell inequality*)

$$-2 \leq S \leq 2 \quad (2.85)$$

where S is given by

$$S = E(\boldsymbol{a},\boldsymbol{b}) - E(\boldsymbol{a},\boldsymbol{b}') + E(\boldsymbol{a}',\boldsymbol{b}) + E(\boldsymbol{a}',\boldsymbol{b}'). \quad (2.86)$$

We will be using this inequality when we discuss Ekert's 1991 protocol (sec 3.5).

Chapter 3

QUANTUM KEY DISTRIBUTION PROTOCOLS

The purpose of quantum key distribution is the *generation* of a secret string of bits which is known only to sending and receiving parties, who initially share no secret information. The key generation involves production of qubits in certain states, followed by randomly determined measurements. A classical, unsecured channel is required for post-processing (sifting, information reconciliation and privacy amplification). Ideally, this string is a perfectly random key, which will typically be used as key to encrypt data on a classical computer. By contrast to ancient methods, the (classical) encrypting and decrypting algorithms are assumed to be known publicly.

QKD protocols are secure even in the existence of an eavesdropper possessing unlimited computational power and utmost quality devices. Unlike the practical classical cipher, which are based on a mixture of a guesswork and mathematics, the unconditional security of these protocols can actually be proved (see 6.2). Secrecy of the key is protected by physical laws.

Today, there are dozens of QKD protocols in the literature. We will be limiting ourselves to popular, discrete [1] protocols.

3.1 BB84

This one is the archetypal QKD protocol, invented by Bennett and Brassard in 1984 [2]. It uses two sets of orthogonal basis pairs, which are usually taken to be X and Z polarization eigenstates of photon.

We adapt the description of the protocol given in [15].

- Alice generates $(4+\delta)n$-bit random string D for data , and $(4+\delta)n$-bit random string B for basis choice.

- Bob generates $(4+\delta)n$-bit random string M to decide whether he will measure the

[1] That is to say, the Hilbert space of states used for encoding information is finite.

incoming qubits in X or Z basis. Basis choice bits decide whether she will use $\{|0\rangle, |1\rangle\}$ or $\{|+\rangle, |-\rangle\}$ basis. The data bit will decide which state will be used, of the chosen basis.

- For each data and basis bit, she prepares a photon in the corresponding state and sends it to Bob over quantum channel.

- Bob measures the incoming photons one by one in accordance with M, and notes down the results.

- Alice announces B over a classical channel. The channel may well be insecure.

- Alice and Bob discard the bits where he measured in a different basis than Alice prepared. Mostly likely [2], there will be at least $2n$ bits left. They start over otherwise.

- Alice picks n bits at random positions from D, tells Bob the positions and values. They compute QBER using these bits to decide whether there was a significant amount of eavesdropping, then throw them away. If QBER is above a certain threshold, they start over. (We will discuss this threshold value when we consider the security conditions in section 6)

- They perform reconciliation, followed by privacy amplification on the remaining n bits, and finally obtain m bits, known to both.

3.2 SARG04

This protocol, introduced by Valerio Scarani, Antonion Acin, Gregoire Ribordy, Nicolas Gisin in 2004 [19] is identical to BB84 except the classical sifting stage. It was designed as a countermeasure to PNS attack (see 2.2.2).

SARG04 involves the following change in BB84. After the transmission, instead of announcing her basis, Alice will instead send a pair of signs $\mathcal{A}_{\omega_x,\omega_z} = \{\omega_x, \omega_z\}$ (each $\in \{+, -\}$) to Bob, which will mean "The state I sent was either $|\omega_x x\rangle$ or $|\omega_z z\rangle$" (one of them

[2] We choose δ such that this probability is acceptable.

denotes the actual state, the other one is picked at random). If Bob, according to his measurement result, can't tell with certainty which is which, they discard the result.

Suppose Alice sent a qubit in $|+x\rangle$ state, and announced the set of possible states as $\mathcal{A}_{++}$, meaning "I sent either $|+x\rangle$ or $|+z\rangle$". If Bob made an X measurement, he would definitely get $+1$, but he cannot be sure because from his perspective, it is also possible that Alice may have sent $|+z\rangle$ and his X measurement yielded $+1$. The situation is symmetric if he performed an Z measurement and got $+1$. However, if he measured Z and got -1, then he definitely is sure that he performed the "wrong" measurement since it not possible for him to get a -1 when Alice sends $|+z\rangle$, thus can conclude that the state Alice sent was $|+x\rangle$. And this happens with $p = 1/4$ probability (unlike BB84 where we had $p = 1/2$). With probability $1 - p = 3/4$, the result is inconclusive and is discarded. [3]

In comparison to BB84, SARG04 halves the key generation rate. However, Eve now needs at least two extra photons (pulses with three or more photons) in order to be sure about in the incoming state without causing any disturbance. This makes two photon states safe, in the sense that they do not risk revealing information to Eve without any disturbance.

Given that there is a non-empty pulse, the probability of getting a pulse with three or more photons is

$$P(n > 2|n > 0) = \frac{1 - P(2) - P(1) - P(0)}{1 - P(0)} \approx \frac{\mu^2}{6} \tag{3.1}$$

For $\mu = 0.12$, this means 0.24% of the non-empty pulses can still be a threat. However, this number is greatly reduced in comparison to BB84.

3.3 B92

Bennett later on realized that quantum key distribution can be made using only 2 arbitrary non-orthogonal states [3]. Although Bennett's original analysis in [3] assumed only non-orthogonality of the basis states $|u_0\rangle$ and $|u_1\rangle$, following [15] we will describe the protocol on a more simple and concrete example. We will take Alice's states to be $|u_0\rangle = |0\rangle$ and $|u_1\rangle = |+\rangle$. The protocol differs from BB84 in encoding and processing photons, so it suffices to describe what happens to a single photon. Alice sends a photon in either $|0\rangle$ or $|+\rangle$ state at random. Bob chooses to measure the incoming photon, again at random, in either X or

[3] The basic idea here is essentially the same used in B92 (section 3.3)

Z basis. By contrast to BB84, he announces his measurement result, but does not reveal his choice of basis.

We see that if his choice of basis was consistent with Alice's, then his measurement result would definitely be $+1$, corresponding to $+1$ eigenstate of X or Z (i.e., $|+\rangle$ or $|0\rangle$). Otherwise, he still may get $+1$ with $1 - p = 1/2$ [4] probability. When, however, he gets -1, he will be certain that his basis is the "wrong" one. And only in this case (which has an overall probability of $(p/2 = 1/4)$ they will keep the result. If we call the bit Alice registers b, Bob should register $\neg b$ [5] since he is certain that he chose the "wrong" basis.

The rest of the protocol is similar to BB84. After they are finished sending/measuring qubits and sifting the results by public discussion, Alice and Bob can apply information reconciliation, followed by privacy amplification to agree on a secret, shared key.

This protocol exploits the indistinguishability of non-orthogonal states (see 2.4.2), while keeping the number of required basis states at minimum.

3.4 Six-state Protocol

Six-state protocol, introduced by D. Bruss, represents the symmetry in Bloch vector representation better by using eigenstates of X, Y, and Z [16]. This time, Alice chooses from 3 possible axes, which will cause them to throw away 2/3 of the bits in the sifting stage, reducing the efficiency in comparison to BB84. The protocol is otherwise identical to BB84. However, the intrinsic symmetry of the protocol simplifies the security analysis and reduces Eve's optimal information gain, increasing the QBER threshold for intercept-resend on all qubits to 33% (which is 25% in BB84) [16].

3.5 E91 (or EPR) Protocol

Unaware of the recent developments, Arthur Ekert re-invented quantum cryptography in 1991 [4]. Unlike the previous protocols however, this new protocol used entangled particles instead of non-orthogonal pairs of state. The security condition was stated in terms of generalized Bell inequality (or CHSH inequality, see 2.4.3) rather than disturbance in the

[4] $p = 1 - |\langle u_0|u_1\rangle|^2$

[5] Here $\neg$ operator is the bitwise-NOT, or base-2 complementary operator. For a single bit, it means $0 \to 1$ and $1 \to 0$.

states. The secrecy is protected by the completeness of quantum mechanics.

The protocol requires a source that emits pairs of entangled particles in singlet state

$$|\Psi^-\rangle = \frac{|0\rangle|1\rangle - |1\rangle|0\rangle}{\sqrt{2}} \tag{3.2}$$

One of the particles is sent to Alice, and the other is to Bob via channel. They both perform measurements in $\boldsymbol{\sigma} \cdot \hat{\boldsymbol{n}}$, where $\hat{\boldsymbol{n}}$ is one of the vectors $\{\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3\}$ for Alice and $\{\boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_3\}$ for Bob, chosen at random for every incoming qubit. These three vectors like in xy plane, and are defined by their azimuthal angles: For Alice $\phi_1^a = 0$, $\phi_2^a = \pi/4$, $\phi_3^a = \pi/2$, for Bob $\phi_1^b = \pi/4$, $\phi_2^b = \pi/2$, $\phi_3^b = 3\pi/4$. Each measurement yields either $+1$ or -1 (we assume that the measurement results are recorded in units of $\hbar/2$).

When the measurement devices of Alice and Bob are aligned, we get a perfect anti-correlation. According to (2.71),

$$E(\boldsymbol{a}_3, \boldsymbol{b}_2) = E(\boldsymbol{a}_2, \boldsymbol{b}_1) = -1 \tag{3.3}$$

$$\tag{3.4}$$

This first set results of aligned measurements can be used to generate the secret key, as in BB84. For other cases, we have

$$E(\boldsymbol{a}_1, \boldsymbol{b}_1) = E(\boldsymbol{a}_3, \boldsymbol{b}_1) = E(\boldsymbol{a}_3, \boldsymbol{b}_3) = -\frac{1}{\sqrt{2}} \tag{3.5}$$

$$E(\boldsymbol{a}_1, \boldsymbol{b}_3) = \frac{1}{\sqrt{2}} \tag{3.6}$$

So, for this configuration, quantum mechanics predicts

$$S = E(\boldsymbol{a}_1, \boldsymbol{b}_1) - E(\boldsymbol{a}_1, \boldsymbol{b}_3) + E(\boldsymbol{a}_3, \boldsymbol{b}_1) + E(\boldsymbol{a}_3, \boldsymbol{b}_3) = -2\sqrt{2} \tag{3.7}$$

and obviously, CHSH inequality (2.85) is violated. That is, of course, assuming that no-one modified the particles along the road.

After the transmission is completed, Alice and Bob can publicly announce their choice of measurement bases (similar to the case in BB84). For cases where their detectors are

not aligned, their measurement results may not agree. However, they can be tested against (3.5) and CHSH inequality to check for eavesdropping.

Note that since the key in *undetermined* before the measurement, it doesn't really matter who creates the entangled pair of qubits. And if the forged qubit pair is not truly entangled, correlation would yield unexpected result, revealing the truth.

We can consider an eavesdropper's intervention by considering a general modification (due to Eve) in correlation terms. Because of Eve's perfect measurements $E(\boldsymbol{a}_i, \boldsymbol{b}_j)$ becomes $E(\boldsymbol{a}_i, \boldsymbol{n}_a)E(\boldsymbol{n}_b, \boldsymbol{b}_j)$ where $\boldsymbol{n}_a$ and $\boldsymbol{n}_b$ are two unit vectors along direction of Eve's detectors [6]. We assume that there is a normalized probability distribution function $\rho(\boldsymbol{n}_a, \boldsymbol{n}_b)$ which describes Eve's strategy. Averaged over this function (which is actually an average over trials, where Eve tries one of the possible configurations with probability $\rho(\boldsymbol{n}_a, \boldsymbol{n}_b)$), sum of new correlation terms becomes

$$\bar{S} = \langle S \rangle_{\rho(\boldsymbol{n}_a, \boldsymbol{n}_b)} = \int d\boldsymbol{n}_a d\boldsymbol{n}_b \rho(\boldsymbol{n}_a, \boldsymbol{n}_b) \times \tag{3.8}$$

$$[(\boldsymbol{a}_1 \cdot \boldsymbol{n}_a)(\boldsymbol{a}_1 \cdot \boldsymbol{n}_b) - (\boldsymbol{a}_1 \cdot \boldsymbol{n}_a)(\boldsymbol{a}_3 \cdot \boldsymbol{n}_b) + (\boldsymbol{a}_3 \cdot \boldsymbol{n}_a)(\boldsymbol{a}_1 \cdot \boldsymbol{n}_b) + (\boldsymbol{a}_3 \cdot \boldsymbol{n}_a)(\boldsymbol{a}_3 \cdot \boldsymbol{n}_b)] \tag{3.9}$$

If we call the azimuthal angles for $\boldsymbol{n}_a$ and $\boldsymbol{n}_b$, α and β respectively, the term between square brackets becomes

$$\cos\alpha \left[\cos\left(\beta - \frac{\pi}{4}\right) - \cos\left(\beta - \frac{3\pi}{4}\right)\right] + \cos\left(\alpha - \frac{\pi}{2}\right) \left[\cos\left(\beta - \frac{\pi}{4}\right) + \cos\left(\beta - \frac{3\pi}{4}\right)\right] = \sqrt{2}\cos(\alpha - \beta) \tag{3.10}$$

and $\bar{S}$ reduces to

$$\bar{S} = -\int d\boldsymbol{n}_a d\boldsymbol{n}_b \rho(\boldsymbol{n}_a, \boldsymbol{n}_b)[\sqrt{2}\boldsymbol{n}_a \cdot \boldsymbol{n}_b] \tag{3.11}$$

For the case of $\boldsymbol{n}_a = -\boldsymbol{n}_b$ for instance, CHSH inequality becomes

$$-\sqrt{2} \leq \bar{S} \leq \sqrt{2} \tag{3.12}$$

We see that this result contradicts (3.7). Furthermore, it does not violate CHSH in-

[6] We will assume that they lie on the xy plane.

equality, which makes it clear that someone has intervened, or as Ekert put it "someone has introduced elements of physical reality".

3.6 Other Protocols

There are plenty of other protocols, built on the ideas of BB84 [16]. Some protocols abondon qubits, and exchange three-level qutrits (or systems with even higher dimensions) instead. Another variation is, Alice and Bob do not choose from X and Z with equal probability. There is also the idea of preparing qubits in superposition of two spatially separated states, one component is then sent to Bob, and only after he recieves it, the second component is sent.

Chapter 4

INFORMATION RECONCILIATION

Even after Alice and Bob complete the sifting stage, due to noise (physical imperfections of the channel, misalignment of polarizers and analyzers, eavesdropping...) in the channel, their keys may not agree. They need to reconcile their keys in order to make them identical. Reconciliation [1] is the procedure of finding and correcting discrepancies between Alice's and Bob's secret keys. This is realized via a (non-secure) classical, public channel.

Before describing the reconciliation protocols, we would like to mention some general, theoretical issues. Throughout the chapter we will model the quantum channel as binary symmetric channel (see 2.1), with bit-flip probability ϵ.

Let us label Alice's sifted key as A, and Bob's as B. They would like to agree on a secret key S of length n after a public discussion on a classical channel. Since anyone can tap into this channel, some bits (which we will call Q) will inevitably be leaked to eavesdropper. They would like to divulge as few as possible during this discussion, however, conditional entropy (2nd term on the right-hand side in (2.13)) gives a lower bound to the least possible number of bits they should exchange on average [2]

$$H(A|B) = nh(\epsilon) \tag{4.1}$$

This is the theoretical lower bound (on average) to the amount of leaked information to Eve during the reconciliation. If we will call the total information leaked $I_E(S|Q)$, on average, they have $n - I_E(S|Q)$ secret bits. We define the number of secret bits per bit in S to be *efficiency of reconciliation* and denote as [3]

[1] Sometimes called information reconciliation or error reconciliation.

[2] Considering the limit of $n \to \infty$ is equivalent to average over trials, since in a very long sequence, the ratios will converge to probabilities by the law of large numbers. This is the statement of preference in [6].

[3] In [6], the following is $-\zeta$. We believe this is a more natural definition.

$$\zeta \equiv 1 - \frac{I_E(S|Q)}{n}. \tag{4.2}$$

This implies that the theoretical bound for efficiency is $1 - h(\epsilon)$, which happens to be the mutual information of Alice and Bob, $H(A : B)$ (see (2.13)). This gives us a way of measuring the "quality" of a given reconciliation protocol.

Reconciliation methods usually have several parameters that depend on channel's error rate, ϵ. A common way of getting an estimate is this. Alice and Bob sacrifice a fraction of their bits (at random positions) by publicly announcing them, and by comparing these bits, they compute ϵ.

4.1 BBBSS

BBBSS is the original reconciliation protocol by Charles H. Bennett, Francois Bessette, Gilles Brassard, Louis Salvail and John Smolin, introduced in 1992 [5], when they announced the first experimental implementation of BB84.

We adopt the description given in [5]. To randomize the positions of errors, Alice and Bob first shuffle their bits in they same way, so that the errors (i.e., bits that do not agree) will hopefully be distributed uniformly throughout A and B. They then break their messages into blocks of k-bits such that each block has about one error. Obviously, the optimal block size depends on the error rate. (We will present a derivation of such a relation when we discuss Cascade in the next section.) They then compute the parity (the summation of a given list of bits, mod 2) for each block and tentatively treat the blocks with matching parity as error-free (they may actually contain even-number of errors since $2N$ is zero in mod 2). For the case of "bad blocks" which contain odd-number of errors, a bisective search is executed (which runs recursively): the block is divided into two sub-blocks and parity is computed for each, block with even number of errors are ignored while others are divided into two blocks... This way, errors are found and corrected. (Corresponding part in Cascade is called `BINARY`, they both perform bisective search except for the part they throw away the last bits of each block they compute the parity.) During the bisective search, an extra $\lceil \log k \rceil$ bits (at most) of parities of sub-blocks is disclosed. In order to avoid leaking extra information to Eve, they thus discard the last bit of each block or sub-block. We see that

key rate decreases with increasing k. This highlights the importance of k.

After this procedure, we expect some errors to remain undetected (in blocks with an even number of error). To remedy the situation, we repeat the whole procedure from the beginning several more times with increasing block sizes, until they are convinced that at most only a few errors remains in their data [4].

To reduce the chances of having errors in the final key, in each iteration, Alice and Bob can compute the parity of a block, made of randomly picked bits. If their blocks are not identical, the parities will disagree with probability 1/2. In that case, they do a bisective search as described above. Assuming that there are l blocks in that pass, the chances of having undetected errors is 2^{-l} with this modification (see `CONFIRM` below).

4.2 Cascade

Gilles Brassard and Louis Salvail have published a reconciliation protocol based on BBBSS in 1993 [6]. By contrast to BBBSS, it does not require discarding the last bit after computing the data block's parity. Instead of throwing them away, they are used for further corrections in following passes.

Following [6], we introduce two useful sub-procedures `BINARY` and `CONFIRM`, before giving a definition of Cascade.

`BINARY`: This is the bisective search part of the protocol, which recursively compares the parities of sub-blocks. This is at the cost of divulging at most $\lceil \log n \rceil$ parity bits to eavesdropper.

1. Alice sends Bob the parity of the first half of her sifted key.

2. Bob computes the parity of the first half of his sifted key and compares to Alice's parity, and decides whether the odd number of errors occurred in the first half or second half.

[4] At some point, the new pass is redundant, as we cannot determine how many passes we actually need beforehand. After a few passes, before starting over, it is possible for them to compare a "brief" hash of their whole data array that divulges very few information of their keys (or a random portion of their keys in large chunks) to get a better idea whether their keys are identical or not (using large chucks is better than using separate small chunks as in [23] in the sense that we can "compress" more with combined data). Afterwards, they may shuffle their data in a way they agree on, and/or throw away a few bits to compensate for the information leak when divulging the hash, and go on.

3. This process is recursively applied to the half determined in step 2, until the error is eventually found. When the error is found, it is corrected by a single bit-flip.

`CONFIRM`: [5] When Alice's and Bob's keys are not identical, this protocols confirms it with probability 1/2. In the absence of errors, it confirms with probability 1.

1. Alice and Bob pick a block at random locations in their keys (they agree on these locations, of course).

2. Alice sends the parity of her subset to Bob.

3. Bob checks whether the parities agree or not. For the case where there are even number of errors, his conclusion will be misleading.

This procedure can be iterated l times, so that failure probability decreases exponentially (2^{-l}).

Cascade: We here give the description of the protocol, followed by a discussion on the optimal choice of block size for each pass —which depends on error rate of the channel— in 4.2.1.

In the first pass (which we will denote as $i = 1$), Alice and Bob split their keys into blocks of k_1 bits. Alice computes and sends the parities of her blocks to Bob. Using `BINARY`, Bob corrects errors in blocks whose parities differ from Alice's corresponding block. So far, the method is very similar to BBBSS.

Beginning from the second pass ($i > 1$) however, we use the following procedure. Alice and Bob construct blocks of k_i bits from their keys [6] (both sides will have n/k_i blocks in total), but this time, the elements of blocks are picked at random. To formalize this, we say we have a function f_i whose domain is $[1 \dots n] \to [1 \dots \lceil n/k_i \rceil]$, that assigns a random block to a given bit index (i.e., it tells which bit is in which block). To refer these blocks in a more compact way, we will make one more definition. K_j^i is a list of bit-positions that defines the jth block of Alice and Bob in ith pass, or in terms of f_i, $K_j^i = \{l | f_i(l) = j\}$.

[5] Although this is not a part of Cascade, we keep this procedure, which originally defined in [6], because it can be incorporated into BBBSS.

[6] They do not work on *copies* of these bits, however. When we say "constructing a block", we merely mean making a list of indices that *point to* these bits.

As in the first pass, Alice sends the parities of the newly formed blocks to Bob. And Bob in return computes the parities of his blocks and executes `BINARY` for differing ones. Let us say the parity for jth block did not match, and after using `BINARY`, they corrected the bit at position l, which is in K_j^i. At this point, Alice and Bob can make use of the previous passes like this. Since the end of each pass every parity mismatch is corrected, this newly found error at l implies that we missed it in all the previous cases. Which means every block that contained lth bit in previous searches had double (or even number of) errors (let us call the set of these blocks $\mathcal{K}$ —this includes all such blocks in the current, and previous passes). Now that we corrected the one at l, they all are left with odd number of errors. We should go back and correct these even "newer" [7] errors! We just pick the smallest block in $\mathcal{K}$[8] and execute `BINARY` on it, this will yield the position of the *other* error we missed, l' and correct it. Let us call the set of blocks containing l', $\mathcal{B}$. Now it happens that not every block containing the lth bit contains l'th, vice and versa. It was the case for the block on which we just used `BINARY` on, but it may not be the case for others in $\mathcal{K}$. There may still be blocks that contained either one of l or l', and upon correction, now left with an odd number of errors. The set of these blocks is, obviously,

$$\mathcal{K}' = \mathcal{B} \nabla \mathcal{K} = (\mathcal{B} \cup \mathcal{K}) \setminus (\mathcal{B} \cap \mathcal{K}). \tag{4.3}$$

If such blocks exists, that is to say, if $\mathcal{K}' \neq \emptyset$, then Bob will find another pair of errors in the same way. This procedure is repeated until there are no more known parity mismatches.

4.2.1 Choosing Good Block Sizes for Cascade

We follow the analysis in [6], which yields the appropriate choice of block size for each pass, such that the probability that the number of errors in a block (say, K_v^1) decreases exponentially with increasing number of passes completed. Let us define $\delta_i(j)$ to be the probability that after the pass $i(\geq 1)$, $2j$ errors remain in K_v^1. The distribution of errors will approximately be binomial distribution, and since both $2j$ and $2j+1$ errors contribute

[7] Of course they were there from the beginning, it's just we haven't noticed them before.

[8] So that the processing will be quick.

to $\delta_1(j)$, we have

$$\delta_1(j) = \binom{k_1}{2j}\epsilon^{2j}(1-\epsilon)^{k_1-2j} + \binom{k_1}{2j+1}\epsilon^{2j+1}(1-\epsilon)^{k_1-(2j+1)} \tag{4.4}$$

And let E_i be the expected number of errors in that block after ith pass (and the goal will become that it is roughly halved after each pass: $E_i = E_{i-1}/2$). For the first pass,

$$E_1 = \sum_{j=1}^{\lfloor k_1/2 \rfloor} 2j \times \delta_1(j) = \epsilon k_1 - \frac{1-(1-2\epsilon)^{k_1}}{2}. \tag{4.5}$$

And let γ_i to be the probability of correcting *at least* 2 errors in ith pass ($i > 1$) for the block K_v^1, which still contains errors even after the corrections in $i-1$th pass. The number of expected errors per block[9] in $i-1$th pass is $\frac{n}{k_1}E_{i-1}$, and the probability that a given (erroneous) bit is in a certain block is k_i/n. Using these, and the fact that errors are corrected in pairs, we can write down the following inequality (for large n)

$$\gamma_i \geq 1 - \left(1 - \left(1 - \frac{k_i}{n}\right)^{\frac{n}{k_1}E_{i-1}}\right)^2 \approx 1 - \left(1 - e^{-k_i E_{i-1}/k_1}\right)^2 \tag{4.6}$$

With the inequality for γ_i, we can give an upper bound for $\delta_i(j)$ (for $i > 1$, of course)

$$\delta_i(j) \leq \left(\sum_{l=j+1}^{\lfloor k_1/2 \rfloor} \delta_{i-1}(l)\right) + \delta_{i-1}(j)(1-\gamma_i) \tag{4.7}$$

If we choose the initial block size k_1 such that

$$\sum_{l=j+1}^{\lfloor k_1/2 \rfloor} \delta_1(l) \leq \frac{1}{4}\delta_1(j) \tag{4.8}$$

and double the block size at each pass (i.e., $k_i = 2k_{i-1}$) and plugging this into (4.6), and using (4.8) within (4.7) in conjunction, we can write

$$\delta_i(j) \leq \frac{1}{4}\delta_{i-1}(j) + \left(1 - \exp\left(2^{i-1}E_{i-1}\right)\right)^2 \delta_{i-1}(j). \tag{4.9}$$

If we furthermore introduce the following restriction

[9]Since our discussion focuses on the error correction in K_v^1, we here refer to the blocks of the first pass.

$$E_1 = ek_1 - \frac{1-(1-2\epsilon)^{k_1}}{2} \leq \frac{\ln 2}{2} \tag{4.10}$$

since $\delta_i(j) \leq \frac{\delta_{i-1}(j)}{2} \leq \frac{\delta_1(j)}{2^{i-1}}$, we have $E_i = \frac{E_{i-1}}{2}$, and thus (4.6) can be written as

$$1 - \gamma_i \leq \left(1 - e^{-2E_1}\right)^2 \leq \frac{1}{4} \tag{4.11}$$

In summary, we choose our "free" parameter k_1 such that it obeys (4.8) and (4.10), and we double it after each pass.

There is, however, no analysis for the number of total passes ω as a function of error rate ϵ (and maybe the length of the sifted key, n), to the best knowledge of the author. Benchmarks in [6] list that $\omega = 4$ is a good choice for $n = 10000$ in a practical weak-pulse implementation with $\mu = 0.12$, within the acceptable range of QBER of BB84.

Chapter 5

PRIVACY AMPLIFICATION

The goal of privacy amplification is to reduce the information of Eve on reconciled bit-string S. One simple way of implementing the idea is this. Alice and Bob can split their keys into blocks of 2 bits, and by XORing these blocks, generate a new key. If Eve had the correct values of bits with 3/4 probability, after XOR, this reduces her chances down to $(3/4)^2 + (1/4)^2 = 10/16$. This is good, but we would like to establish an exponentially decreasing upper bound for Eve's information.

5.1 Privacy Amplification Using Hash Functions

Following [13], we will show that if Eve's information on S is no more than t deterministic bits, when a randomly and public chosen hash function $h : \{0,1\}^n \to \{0,1\}^r$ (where $r = n - t - s$) maps S into a smaller key K on which Eve's information is less than $2^{-s}/\ln 2$ bits, where s is called the security parameter. The key after privacy amplification is it's final form, so we expect Eve's information to be exponentially small at this point, which is satisfied by this method.

When we say hash function in this context, we actually mean linear universal$_2$ (we will just say universal) functions $\{0,1\}^n \to \{0,1\}^r$. Universal functions are a class of functions $\mathcal{G}$ which are $\mathcal{A} \to \mathcal{B}$, and for all any distinct x_1 and x_2, the probability for $g(x_1) = g(x_2)$ is at most $1/|\mathcal{B}|$ (where $|\mathcal{B}|$ is the number of elements in $\mathcal{B}$ —for our case, it is 2^r) when the function g is chosen from $\mathcal{G}$ using a uniform distribution. Such linear universal functions can be described using $r \times n$ matrices with 0s and 1s as its elements ($GF(2)$).

Collision probability for a random variable X over an alphabet $\mathcal{X}$, with a probability distribution is defined to be

$$P_c(X) = \sum_{x \in \mathcal{X}} P_X(x)^2. \tag{5.1}$$

The Rényi entropy (of order two) of X is defined as

$$R(X) = -\log P_c(X). \tag{5.2}$$

The conditional entropy is extended in a similar manner as in the case of Shannon's entropy

$$R(X|Y) = \sum_{y \in \mathcal{Y}} P_Y(y) R(X|Y = y). \tag{5.3}$$

At this point, we will state *Jensen's inequality*, which says, for a concave function $f(x)$ (such as logarithm function)

$$f\left(\sum p_i x_i\right) \geq \sum p_i f(x_i) \tag{5.4}$$

when $\sum p_i = 1$, which is the case for probabilities [1]. In other words,

$$f(\langle x_i \rangle) \geq \langle f(x_i) \rangle \tag{5.5}$$

Since $H(X) = -\langle \log P_X(X) \rangle$ and $R(X) = -\log \langle P_X(X) \rangle$, we see that Rényi entropy is bounded by Shannon entropy

$$R(X) \leq H(X). \tag{5.6}$$

By the same token, for conditional entropies, we can easily derive

$$R(X|Y) \leq H(X|Y). \tag{5.7}$$

Before analyzing Eve's information, we will derive one other inequality, which is not as straightforward as the ones so far:

$$H(K|G) \geq R(K|G) \geq r - \frac{2^{r-R(X)}}{\ln 2} \tag{5.8}$$

where G is the (uniformly) random choice of a universal hash function, and $K = G(X)$. We begin with explicitly writing out the Rényi entropy

[1] For any pair of positive real numbers that obey $p_1 + p_2 = 1$, concavity implies $f(p_1 x_1 + p_2 x_2) \geq p_1 f(x_1) + p_2 f(x_2)$ for any x_1 and x_2, and by induction, this can be generalized to any number of variables.

$$R(G(X)|G) = \sum_g P_G(g) R(G(X)|G=g) = \sum_g P_G(g)(-\log P_c(G(X)|G=g)) \tag{5.9}$$

Using Jansen's inequality on the last expression, we have

$$R(G(X)|G) \geq -\log\left(\sum_g P_G(g) P_c(G(X)|G=g)\right) \tag{5.10}$$

The sum inside the logarithm is equal to the probability for $g(x_1) = g(x_2)$ when g is randomly chosen according to P_G and x_1, x_2 are randomly and independently chosen according to P_X), that is to say,

$$\Pr[G(X_1) = G(X_2)]. \tag{5.11}$$

We can also write the same thing this way

$$\Pr[X_1 = X_2] + \Pr[X_1 \neq X_2]\Pr[G(X_1) = G(X_2)|X_1 \neq X_2] \tag{5.12}$$

which makes a direct reference to the definition of a universal hash function. The chances for $G(X_1) = G(X_2)$ when $X_1 \neq X_2$ is 2^{-r}, and introducing the collision entropy, we can write down the following inequality

$$\Pr[X_1 = X_2] + \Pr[X_1 \neq X_2]\Pr[G(X_1) = G(X_2)|X_1 \neq X_2] \leq P_c(X) + (1 - P_c(x))2^{-r}. \tag{5.13}$$

An even higher bound is

$$P_c(X) + (1 - P_c(x))2^{-r} < 2^{-R(X)} + 2^{-r} = 2^{-r}\left(1 + 2^{-r-R(X)}\right). \tag{5.14}$$

Plugging this back into the logarithm, we have

$$H(K|G) \geq R(K|G) \geq r - \log(1 + 2^{-r+R(X)}) \tag{5.15}$$

Using the bound for logarithm $\log(1+x) \leq x/\ln 2$, we recover the original inequality (5.8). We now have the tools to proceed to derive an upper bound to Eve's information.

Suppose Eve has a way of obtaining t deterministic bits from the reconciled n-bit key S, which we will represent using the function $e(S)$, $e : \{0,1\}^n \to \{0,1\}^t$ $(t < n)$. The nature of the function is determined by the eavesdropping strategy used by Eve. Let $v \in \{0,1\}^t$ to be a particular value of V, as result of Eve's measurements. There will be many possible s that satisfies $e(s) = v$, only one being the correct S, and Eve will have to choose among them. Let the number of possible choices for s be c, then $P_{S|V=v} = 1/c$ and

$$P_c(S|V = v) = c\frac{1}{c^2} = \frac{1}{c} \tag{5.16}$$

$$R(S|V = v) = \log c \tag{5.17}$$

When Eve's Rényi entropy $R(S|V = v)$ is known to be (at least) c, we can also write

$$H(K|G, V = v) \geq r - \log\left(1 + 2^{r-c}\right) \geq r - \frac{2^{r-c}}{\ln 2} \tag{5.18}$$

Averaging this over the possible values of v, we derive $H(K|GV)$. , we obtain the following for Eve's information on the key

$$I(K; GV) = H(K) - H(K|GV) \leq r - \sum_v P_V(v) H(K|G, V = v) \tag{5.19}$$

Using $P_V(v) = c2^{-n}$

$$I(K; GV) \leq \sum_v c2^{-n} \frac{2^r}{c \cdot \ln 2} = \frac{2^{-n+t+r}}{\ln 2} = \frac{2^{-s}}{\ln 2} \tag{5.20}$$

which establishes the exponentially decreasing upper bound for Eve's information, in terms of the security parameter s. It is normally used to define the value of $r = n - s - t$, for a pre-set value of n and t.

Choosing the class of linear universal hash functions (which can be represented by $n \times r$ matrices of bits) as $\mathcal{G}$ has a practical downside, however. It requires parties to generate nr random bits to transmit over a classical line, a number which can grow very fast. One can however reduce this requirement, using Toeplitz matrices [21]. A Toeplitz matrix satisfies $M_{ij} = M_{i+\delta,j+\delta}$ for $1 \leq i, i + \delta \leq r$ and $1 \leq j, j + \delta \leq n$ (i.e., it has the same number along a diagonal). It has only $n + r - 1$ independent terms, which is a feasible number of bits to

transfer over a classical channel. [2]

Explicitly, if we have arrays of random bits $\{a_1, a_2, a_3, \ldots a_n\}$ and $\{b_2, b_3, \ldots b_m\}$ for instance, we can construct the following Toeplitz matrix

$$M = \begin{pmatrix} a_1 & a_2 & a_3 & \ldots & \ldots & a_n \\ b_2 & a_1 & a_2 & \ddots & & \vdots \\ b_3 & b_2 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{n-r+2} & a_{n-r+3} \\ \vdots & & \ddots & b_2 & a_{n-r+1} & a_{n-r+2} \\ b_r & \ldots & \ldots & b_3 & b_2 & a_{n-r+1} \end{pmatrix} \tag{5.21}$$

[2] We will not try to establish that Toeplitz matrices are universal here, as it does not fall into the scope of this text. Keen reader can, however, follow the references in [21].

Chapter 6

SECURITY

In this chapter we will give a brief introduction to security related issues and cite some important results in QKD for the sake of completeness. We will omit the proof for unconditional security by Shor and Preskill —which otherwise requires us to broaden the background information to include linear error correction codes and CSS codes— and cite it instead. We will discuss the basic ideas involved, however.

6.1 Random Number Generation

The encryption techniques usually require perfectly random numbers, which cannot be generated using classical methods. Random number sequences generated by classical computers use an algorithm that relates a given number to another deterministically, with a single function call as the interface

$$x_{n+1} = f(x_n) \tag{6.1}$$

Although the sequence $\{x_1, x_2, ..., x_N\}$ may look random, each number is directly related to the previous one. The initial number in the sequence (sometimes called the seed) is usually derived from computer's clock at function's execution time, so that the sequence will mostly be different each time the program is run. It is possible to update the seed value using current system time and maybe the previous random number before each call to increase randomness, but the whole thing is deterministic by construction. These functions are called *pseudo-random number generators*, and because the numbers are correlated in a given sequence, they are avoided when security is of major concern.

6.2 QBER Threshold for Secure Key

We have already noted that when Eve measures every incoming qubit in X or Z and sends a qubit in accordance to her measurement (this is called the intercept-resend strategy), she will introduce a QBER of 0.25. Following the analysis in [12], we will not restrict Eve to use X and Z axes, but allow for an arbitrary tilt θ for her measurement bases.

Let us say Alice encoded her qubit it Z basis. The probability for Eve to get the "correct" result is

$$p_z(\theta) = \cos^2\left(\frac{\theta}{2}\right) = \frac{1}{2} + \frac{\cos\theta}{2} \tag{6.2}$$

Only the angles *between* measurement basis and encoding basis matters, and since $Z \to X$ means a $\theta = \pi/2$ rotation in Bloch sphere, we can work out the case for X by a simple substitution $\theta \to \pi/2 - \theta$: $p_x(\theta) = p_z(\pi/2 - \theta)$ Since X and Z basis are chosen with equal likelihood, the probability for Eve to get the "correct" result is

$$p^{(r)}(\theta) = \frac{1}{2}p_x(\theta) + \frac{1}{2}p_z(\theta) = \frac{1}{2} + \frac{\cos\theta + \sin\theta}{2} \tag{6.3}$$

where the superscript (r) denotes this is the probability associated with the raw (prior to sifting) key. Since the quantum channel is a binary channel, the information of Eve will be given by

$$I^{(r)}(\theta) = 1 - h\left(p^{(r)}(\theta)\right) \tag{6.4}$$

After the bases are announced, Eve will know which basis is used for each qubit. She can tell whether a bit was encoded in X or Z basis now, so she can separate her bits into two groups. This means, Eve effectively behaves like two receivers listening to channel for X bits only and channel for Z bits only. As a result, the information gain of Eve after sifting will be equal to the average information gained by these two imaginary listeners

$$I^{(s)}(\theta) = \frac{1}{2}I_z^{(s)}(\theta) + \frac{1}{2}I_x^{(s)}(\theta) \tag{6.5}$$

where

$$I_z^{(s)}(\theta) = 1 - h\left(p_z(\theta)\right) \tag{6.6}$$

$$I_x^{(s)}(\theta) = 1 - h\left(p_x(\theta)\right) \tag{6.7}$$

We see that Eve's information on the raw key is maximized when $\theta = \pi/4$, whereas for the sifted key, the ideal angle is $\theta = 0$.

Eve introduces an average error probability of

$$\frac{1}{2}(1 - p_z(\theta)) + \frac{1}{2}(1 - p_x(\theta)) \tag{6.8}$$

which is maximized for $\theta = \pi/4$ (this configuration is called *Breidbart basis*), becoming $\frac{1-1/\sqrt{2}}{2} \approx 0.15$. So, when Eve is limited to attach photons one by one (individual attack), the threshold value for QBER is 0.15

We will state a theorem from [16] for the ultimate security, which is valid even when Eve has a way of manipulating all qubits jointly after the public announcement, without giving the proof. Alice and Bob can establish a secure key using reconciliation and privacy amplification if and only if $I(A : B) \geq I(A : E)$. $I(A : B)$ is readily $1/2$ for the case of BB84, so this condition becomes

$$\frac{1}{2} \geq 1 - h(\epsilon), \tag{6.9}$$

or numerically, $\epsilon < 0.11$, which defines the threshold for acceptable QBER for secure key.

We would like to emphasize that these proofs assume ideal conditions. Flaws of any kind, such as weak pulses or detectors efficiency mismatches, should be taken into account when giving a threshold value for QBER [16].

Chapter 7

RESULTS AND CONCLUSION

7.1 Simulation of QKD

We have implemented a computer simulation of BB84, which includes QBER estimation, reconciliation (Cascade) and privacy amplification (Toeplitz matrices) in Go programming language [1]. We chose Go because the language has a natural representation of communication channels as a native type. The program runs the code for Alice and Bob (and Eve) in parallel in separate goroutines. Although the whole code appears to be monolithic, breaking the code into submodules should be straightforward since data structures were defined that way. Also, the program can easily be split into separate programs, which communicate over network, by using a module such as JSON RPC. Since we are only concerned with the results of simulation, we did not take these steps.

We compiled the program on an x86-64 machine, using the standard (non-GCC) compiler

```
$ 6g bb84.go && 6l -o bb84 bb84.6
```

The parameters, their defaults values and functions, accepted by the program, are as follows.

```
-n=100: Number of qubits to be generated
-s=0: Security parameter for Cascade
-r=0.5: Ratio of bits to be used for QBER estimation
-v=false: Verbose output  (too much output)
-S="intercept-resend": Strategy for Eve: nil, intercept-resend, breidbard
-T=1: The probability for Eve to tamper with the qubit
-k1=73: Cascade parameter: initial block size
```

[1] We would like to note that the language is still under development. We compiled the source with version 6063 of the Go compiler. Possible changes required to compile the code in future releases will be listed in the Release History page in Go web-site: http://golang.org/doc/devel/release.html

Following is the output of a sample run the simulation. Eve uses intercept-resend attack 44% of incoming qubits, gaining partial information on the key without disturbing the protocol. The security parameter is chosen to be $s = 10$, so that Eve's information is next to nil.

```
./bb84  -n 100 -k1 4 -s 10 -T 0.44

QBER was estimated to be 0.07142857142857142
We got 29 bits out of BB84
Starting reconciliation
Divulged: 4
15 bits final key:
Alice: [0 1 1 0 1 0 0 1 1 0 1 0 1 1 1]
Bob  : [0 1 1 0 1 0 0 1 1 0 1 0 1 1 1]
```

(We picked $n = 100$ and disabled verbose mode to avoid a lengthy output.) We see that out of 100 qubits, Alice and Bob can extract 29 bits of "sifted" key: sifting procedure recudes the raw key length to half, and half[2] of the sifted keys are used for QBER estimation.

When we let Eve to tamper with qubits each time, we get QBER=0.25, thus Alice and Bob cancel the protocol.

```
./bb84  -n 10000 -k1 4 -s 10 -T 1

QBER was estimated to be 0.25273390036452004
We got 2469 bits out of BB84
Someone is badly tampering with the connection. Quiting.
```

[2]This behavior can be changed with `-r` parameter.

Appendix A

GO CODE TO SIMULATE BB84

```
1  package main
2
3  import (
4          "math"
5          "cmath"
6          "rand"
7          "flag"
8          "log"
9          "time"
10 )
11
12 const QBERThreshold = 0.11
13
14 var nraw = flag.Int("n", 100, "Number of qubits to be generated")
15 var verbose = flag.Bool("v", false, "Verbose output (too much output)")
16 var qberBits = flag.Float64("r", 0.5,
17         "Ratio of bits to be used for QBER estimation")
18 // The equations for k1 are pretty complicated to be solved in Go.
19 // One can pipe through math(1) of Mathematica, or generate a look-up table
20 // instead.
21 // We expect user to input these numbers beforehand
22 var k1 = flag.Uint("k1", 73, "Cascade parameter: initial block size")
23 var strategy = flag.String("S", "intercept-resend",
24         "Strategy for Eve: nil, intercept-resend, breidbard")
25 var s = flag.Uint("s", 0, "Security parameter for Cascade")
26 var eveTamperP = flag.Float64("T", 1.0,
27         "The probability for Eve to tamper with the qubit")
28 // ========================
29
30 // Current version does not allow casting from bool
31 func Bool2Int(v bool) int {
```

```
32          if v == true {
33                  return 1
34          }
35          return 0
36  }
37
38  // Workaround. Print function outputs true/false with a bool array.
39  func BoolArray2IntArray(v []bool) []int {
40          out := make([]int, len(v))
41          for i := 0; i < len(v); i++ {
42                  out[i] = Bool2Int(v[i])
43          }
44          return out
45  }
46
47  func RandomBit() bool {
48          return rand.Float() < 0.5
49  }
50
51  func RandomBitArray(n int) []bool {
52          a := make([]bool, n)
53          for j := 0; j < n; j++ {
54                  a[j] = RandomBit()
55          }
56          return a
57  }
58
59  // ============================
60
61  type Qubit struct {
62          a, b complex128
63  }
64
65  func cdiv(c complex128, d float64) complex128 {
66          return cmplx(real(c)/d, imag(c)/d)
67  }
68
69  func (q *Qubit) normalize() {
```

```
70          n := math.Sqrt(real(q.a*cmath.Conj(q.a) + q.b*cmath.Conj(q.b)))
71          q.a = cdiv(q.a, n)
72          q.b = cdiv(q.b, n)
73  }
74
75  func (q *Qubit) InitC(a, b complex128) {
76          q.a = a
77          q.b = b
78          q.normalize()
79  }
80
81  func (q *Qubit) InitR(a, b float64) {
82          q.a = cmplx(a, 0)
83          q.b = cmplx(b, 0)
84          q.normalize()
85  }
86
87  // Returns <q|p>
88  func Inner(q, p *Qubit) complex128 {
89          return cmath.Conj(q.a)*p.a + cmath.Conj(q.b)*p.b
90  }
91
92  func (q *Qubit) Measure(m *Qubit) (res bool) {
93          A := Inner(m, q)
94          P := real(cmath.Conj(A) * A)
95          res = rand.Float64() < P
96          if A != 0 {
97                  q.a = m.a
98                  q.b = m.b
99          }
100         return
101 }
102
103 func NewQubit(base, data bool) Qubit {
104         var q Qubit
105         if base { // computational basis
106                 if data { // |0>
107                         q.InitR(1, 0)
```

```
108                 } else {
109                         q.InitR(0, 1)
110                 }
111         } else {
112                 if data { // |+>
113                         q.InitR(1, 1)
114                 } else {
115                         q.InitR(1, -1)
116                 }
117         }
118
119         return q
120 }
121
122 // ================================
123
124 type QKD struct {
125         n       int
126         data    []bool
127         base    []bool
128         sifted  []bool
129         name    string
130         qber    float64
131         qch     chan Qubit  // Channel to send qubits over
132         sch     chan []bool // Channel used in sifting
133         permch  chan []int  // Used in computing QBER
134         // Only for Eve
135         qchB    chan Qubit  // Channel to send qubits over
136         schB    chan []bool // Channel used in sifting
137         permchB chan []int  // Used in computing QBER
138 }
139
140 func NDifferingBits(a, b []bool) int {
141         if len(a) != len(b) {
142                 panic("Should not happen")
143         }
144         ndiff := 0
145         for i := 0; i < len(a); i++ {
```

```
146                 if a[i] != b[i] {
147                         ndiff++
148                 }
149         }
150         return ndiff
151 }
152
153 func (e *QKD) Init() {
154         e.data = RandomBitArray(e.n)
155         e.base = RandomBitArray(e.n)
156 }
157
158 func (alice *QKD) Alice() {
159         for i := 0; i < alice.n; i++ {
160                 q := NewQubit(alice.base[i], alice.data[i])
161                 //alice.print("Sending qubit #",i,":", q)
162                 alice.qch <- q
163         }
164         alice.sch <- alice.base
165         bobsBase := <-alice.sch
166
167         alice.print("Base:", BoolArray2IntArray(alice.base))
168         alice.print("Data:", BoolArray2IntArray(alice.data))
169
170         sifted := make([]bool, 0, alice.n)
171         for i := 0; i < alice.n; i++ {
172                 if alice.base[i] == bobsBase[i] {
173                         l := len(sifted)
174                         sifted = sifted[0 : l+1]
175                         sifted[l] = alice.data[i]
176                 } else {
177                         alice.print("Bases do not match for qubit #",
178                                 i, ", throwing away")
179                 }
180         }
181         alice.print("Sifted(1):", BoolArray2IntArray(sifted))
182
183         perm := rand.Perm(len(sifted))
```

```
184         alice.permch <- perm
185         nQberEstimationBits := int(float64(len(perm)) * *qberBits)
186         permEstimate := perm[0:nQberEstimationBits]
187         permRemaining := perm[nQberEstimationBits:]
188         qberEstimateBits := make([]bool, len(permEstimate))
189         for i := 0; i < len(qberEstimateBits); i++ {
190                 qberEstimateBits[i] = sifted[perm[i]]
191         }
192         alice.sch <- qberEstimateBits
193         bobsQberEstimateBits := <-alice.sch
194         ndiff := NDifferingBits(qberEstimateBits, bobsQberEstimateBits)
195         alice.qber = float64(ndiff) / float64(len(qberEstimateBits))
196
197         remainingBits := make([]bool, len(permRemaining))
198         for i := 0; i < len(remainingBits); i++ {
199                 remainingBits[i] = sifted[perm[i]]
200         }
201
202         alice.sifted = remainingBits
203
204         alice.print("Sifted(2):", BoolArray2IntArray(alice.sifted))
205 }
206
207 func (bob *QKD) Bob() {
208         for i := 0; i < bob.n; i++ {
209                 q := <-bob.qch
210                 m := NewQubit(bob.base[i], true)
211                 bob.data[i] = q.Measure(&m)
212         }
213         alicesBase := <-bob.sch
214         bob.sch <- bob.base
215
216         bob.print("Base:", BoolArray2IntArray(bob.base))
217         bob.print("Data:", BoolArray2IntArray(bob.data))
218
219         sifted := make([]bool, 0, bob.n)
220         for i := 0; i < bob.n; i++ {
221                 if bob.base[i] == alicesBase[i] {
```

```
222                         l := len(sifted)
223                         sifted = sifted[0 : l+1]
224                         sifted[l] = bob.data[i]
225                 } else {
226                         bob.print("Bases do not match for qubit #",
227                                 i, ", throwing away")
228                 }
229         }
230
231         bob.print("Sifted(1):", BoolArray2IntArray(sifted))
232
233         perm := <-bob.permch
234         nQberEstimationBits := int(float64(len(perm)) * *qberBits)
235         permEstimate := perm[0:nQberEstimationBits]
236         permRemaining := perm[nQberEstimationBits:]
237         qberEstimateBits := make([]bool, len(permEstimate))
238         for i := 0; i < len(qberEstimateBits); i++ {
239                 qberEstimateBits[i] = sifted[perm[i]]
240         }
241         alicesQberEstimateBits := <-bob.sch
242         bob.sch <- qberEstimateBits
243         ndiff := NDifferingBits(qberEstimateBits, alicesQberEstimateBits)
244         bob.qber = float64(ndiff) / float64(len(qberEstimateBits))
245
246         remainingBits := make([]bool, len(permRemaining))
247         for i := 0; i < len(remainingBits); i++ {
248                 remainingBits[i] = sifted[perm[i]]
249         }
250
251         bob.sifted = remainingBits
252
253         bob.print("Sifted(2):", BoolArray2IntArray(bob.sifted))
254 }
255
256 func (eve *QKD) Eve() {
257         for i := 0; i < eve.n; i++ {
258                 q := <-eve.qch
259                 if rand.Float64() < *eveTamperP {
```

```
260                         switch *strategy {
261                         case "nil":
262
263                         case "intercept-resend":
264                                 m := NewQubit(eve.base[i], true)
265                                 eve.data[i] = q.Measure(&m)
266
267                         case "breidbart":
268                                 var m Qubit
269                                 m.InitR(math.Cos(math.Pi/8),
270                                                 math.Sin(math.Pi/8))
271                                 eve.data[i] = q.Measure(&m)
272
273                         default:
274                                 log.Exit("Unknown strategy for Eve")
275                         }
276                 }
277                 eve.qchB <- q
278         }
279         alicesBase := <-eve.sch
280         eve.schB <- alicesBase
281
282         bobsBase := <-eve.schB
283         eve.sch <- bobsBase
284
285         perm := <-eve.permch
286         eve.permchB <- perm
287
288         alicesQberEstimateBits := <-eve.sch
289         eve.schB <- alicesQberEstimateBits
290
291         bobsQberEstimateBits := <-eve.schB
292         eve.sch <- bobsQberEstimateBits
293 }
294
295 func (e *QKD) Sifted() []bool {
296         return e.sifted
297 }
```

```
298
299 func (e *QKD) QBER() float64 {
300         return e.qber
301 }
302
303 func (e *QKD) print(v ...interface{}) {
304         if *verbose == false {
305                 return
306         }
307         log.Stderr(e.name, v)
308 }
309
310 // ================================
311
312 // Some constants for Cascade
313 const MAXPASS = 100
314 const NMINPASS = 2
315
316 // A block is the set of indices, referring to the position in data array
317 type Block []int
318 // A pass is a set of blocks, that covers whole data array
319 type Pass []Block
320
321
322 func (b *Block) Parity(d []bool) bool {
323         sum := 0
324         for _, j := range *b {
325                 //      log.Stderr("data index: ", j)
326                 sum += Bool2Int(d[j])
327         }
328
329         return sum&1 == 1
330 }
331
332 func (pass *Pass) whichBlock(index int) int {
333         for blocki, block := range *pass {
334                 for _, j := range block {
335                         if j == index {
```

```
336                                 return blocki
337                         }
338                 }
339         }
340
341         panic("Should never happen")
342         return -1
343 }
344
345 func newPass(n, blocksize int, randomized bool) Pass {
346         var perm []int
347         if randomized {
348                 perm = rand.Perm(n)
349         } else {
350                 perm = make([]int, n)
351                 for j := 0; j < n; j++ {
352                         perm[j] = j
353                 }
354         }
355
356         nfullblocks := n / blocksize
357         remainder := n - nfullblocks*blocksize
358         nblocks := n/blocksize + Bool2Int(remainder != 0)
359
360         p := make(Pass, nblocks)
361
362         for j := 0; j < nfullblocks; j++ {
363                 p[j] = perm[j*blocksize : (j+1)*blocksize]
364         }
365
366         if remainder != 0 {
367                 p[nfullblocks] = perm[nfullblocks*blocksize:]
368         }
369
370         return p
371 }
372
373 func (alice *Cascade) Alice() {
```

```
374         for ; ; alice.i++ {
375                 alice.print("Started pass #", alice.i)
376                 pass := <-alice.chPass
377                 alice.print("Received indices")
378                 alice.addPass(pass)
379                 // number of corrected blocks in the current pass
380                 ncorrected := 0
381                 for blocki, block := range pass {
382                         alicesParity := block.Parity(alice.d)
383                         alice.chParity <- alicesParity
384                         bobsParity := <-alice.chParity
385
386                         if alicesParity != bobsParity {
387                                 alice.print("Parities for block #", blocki,
388                         "do not match, starting error correction")
389                                 alice.correctBlockAlice(alice.i, blocki)
390                                 ncorrected++
391                         }
392                 }
393
394                 if ncorrected == 0 && alice.i >= NMINPASS {
395                         alice.print("Bye!")
396                         break
397                 }
398         }
399 }
400
401 func (bob *Cascade) Bob() {
402         for ; ; bob.i++ {
403                 bob.print("Started pass #", bob.i)
404                 pass := newPass(len(bob.d), bob.blocksize<<uint(bob.i),
405                                         bob.i != 0)
406                 bob.chPass <- pass
407                 bob.print("Created and sent the indices")
408                 bob.addPass(pass)
409
410                 // number of corrected blocks in the current pass
411                 ncorrected := 0
```

```
412                 for blocki, block := range pass {
413                         alicesParity := <-bob.chParity
414                         bobsParity := block.Parity(bob.d)
415                         bob.chParity <- bobsParity
416                         if alicesParity != bobsParity {
417                                 bob.print("Parities for block #", blocki,
418                         "do not match, starting error correction")
419                                 bob.correctBlockBob(bob.i, blocki)
420                                 ncorrected++
421                         }
422                 }
423
424                 if ncorrected == 0 && bob.i >= NMINPASS {
425                         bob.print("Bye!")
426                         break
427                 }
428         }
429 }
430
431 func (c *Cascade) correctBlockBob(i, blocki int) {
432         block := c.k[i][blocki]
433         ix := c.binaryBob(block)
434         for j := 0; j < i; j++ {
435                 pass := c.k[j]
436                 bi := pass.whichBlock(ix)
437                 c.correctBlockBob(j, bi)
438         }
439 }
440
441 func (c *Cascade) correctBlockAlice(i, blocki int) {
442         block := c.k[i][blocki]
443         ix := c.binaryAlice(block)
444         for j := 0; j < i; j++ {
445                 pass := c.k[j]
446                 bi := pass.whichBlock(ix)
447                 c.correctBlockAlice(j, bi)
448         }
449 }
```

```
450
451 func (bob *Cascade) binaryBob(block Block) int {
452         if len(block) == 1 {
453                 bob.d[block[0]] = !bob.d[block[0]]
454                 bob.print("Error at bit #", block[0], "has been corrected")
455                 return block[0]
456         }
457
458         l := len(block)
459         b0 := block[0 : l/2]
460         b1 := block[l/2:]
461
462         bobsParity := b0.Parity(bob.d)
463         alicesParity := <-bob.chParity
464         bob.chParity <- bobsParity
465         if bobsParity != alicesParity {
466                 return bob.binaryBob(b0)
467         } else {
468                 return bob.binaryBob(b1)
469         }
470
471         panic("Should never happen")
472         return -1
473 }
474
475 func (alice *Cascade) binaryAlice(block Block) int {
476         if len(block) == 1 {
477                 return block[0]
478         }
479
480         l := len(block)
481         b0 := block[0 : l/2]
482         b1 := block[l/2:]
483
484         alicesParity := b0.Parity(alice.d)
485         alice.chParity <- alicesParity
486         bobsParity := <-alice.chParity
487
```

```
488         alice.divulged++
489
490         if bobsParity != alicesParity {
491                 return alice.binaryAlice(b0)
492         } else {
493                 return alice.binaryAlice(b1)
494         }
495
496         panic("Should never happen")
497         return -1
498 }
499
500 func (c *Cascade) print(v ...interface{}) {
501         if *verbose == false {
502                 return
503         }
504         log.Stderr(c.name, v)
505 }
506
507 func (c *Cascade) addPass(pass Pass) {
508         l := len(c.k)
509         c.k = c.k[0 : l+1]
510         c.k[l] = pass
511 }
512
513 func (c *Cascade) Init() {
514         c.k = make([]Pass, 0, MAXPASS)
515 }
516
517 func (c *Cascade) Divulged() int {
518         return c.divulged
519 }
520
521 func (c *Cascade) Data() []bool {
522         return c.d
523 }
524
525 type Cascade struct {
```

```
526         k           []Pass // Pass data, for each pass
527         divulged    int
528         d           []bool // Data array
529         i           int    // Current pass #
530         chParity    chan bool
531         chBINARY    chan int
532         chPass      chan Pass
533         ch          chan int // Auxillary channel
534         blocksize   int
535         name        string
536 }
537
538 // ====================================
539
540 type Toeplitz struct {
541         data []bool
542         ch   chan []bool
543         m    [][]bool
544         s, t int
545 }
546
547 func (alice *Toeplitz) Alice() {
548         n := len(alice.data)
549         r := n - alice.s - alice.t
550         if r <= 0 {
551                 panic("Eek")
552         }
553         a := RandomBitArray(n)
554         b := RandomBitArray(r)
555
556         alice.ch <- a
557         alice.ch <- b
558
559         alice.m = make([][]bool, r)
560         for i := 0; i < r; i++ {
561                 row := make([]bool, n)
562                 for j := 0; j < n; j++ {
563                         for k := 0; k < i; k++ {
```

```
564                                     row[k] = b[k]
565                             }
566                             for k := i; k < n-i; k++ {
567                                     row[k] = a[k]
568                             }
569                     }
570                     alice.m[i] = row
571             }
572
573             alice.data = BinaryMatrixTimesVector(alice.m, alice.data)
574     }
575
576     func (bob *Toeplitz) Bob() {
577             n := len(bob.data)
578             r := n - bob.s - bob.t
579             if r <= 0 {
580                     panic("Eek")
581             }
582
583             a := <-bob.ch
584             b := <-bob.ch
585
586             bob.m = make([][]bool, r)
587             for i := 0; i < r; i++ {
588                     row := make([]bool, n)
589                     for j := 0; j < n; j++ {
590                             for k := 0; k < i; k++ {
591                                     row[k] = b[k]
592                             }
593                             for k := i; k < n-i; k++ {
594                                     row[k] = a[k]
595                             }
596                     }
597                     bob.m[i] = row
598             }
599
600             bob.data = BinaryMatrixTimesVector(bob.m, bob.data)
601     }
```

```
602
603 func BinaryMatrixTimesVector(m [][]bool, v []bool) []bool {
604         res := make([]bool, len(m))
605         for i, row := range m {
606                 if len(row) != len(v) {
607                         panic("Eek!")
608                 }
609                 sum := 0
610                 for j := 0; j < len(row); j++ {
611                         // Can use XOR and AND here.
612                         sum += Bool2Int(row[j]) * Bool2Int(v[j])
613                 }
614                 res[i] = sum&1 == 1
615         }
616         return res
617 }
618
619 // ================================================
620
621 func init() {
622         flag.Parse()
623         rand.Seed(time.Seconds())
624 }
625
626 func main() {
627         // BB84 + QBER Estimation
628         // channels between Alice and Eve
629         qch1 := make(chan Qubit)
630         sch1 := make(chan []bool)
631         permch1 := make(chan []int)
632
633         // channels between Eve and Bob
634         qch2 := make(chan Qubit)
635         sch2 := make(chan []bool)
636         permch2 := make(chan []int)
637
638         alice := &QKD{n: *nraw, qch: qch1, sch: sch1, permch: permch1,
639                 name: "Alice:"}
```

```
640         bob := &QKD{n: *nraw, qch: qch2, sch: sch2, permch: permch2,
641                 name: "Bob   :"}
642         eve := &QKD{n: *nraw, qch: qch1, sch: sch1, permch: permch1,
643                 qchB: qch2, schB: sch2, permchB: permch2, name: "Eve   :"}
644
645         alice.Init()
646         bob.Init()
647         eve.Init()
648
649         go bob.Bob()
650         go eve.Eve()
651         alice.Alice()
652
653         log.Stderr("QBER was estimated to be", bob.QBER())
654         log.Stderr("We got", len(alice.Sifted()), "bits out of BB84")
655
656         if bob.QBER() > QBERThreshold {
657                 log.Exit(
658                 "Someone is badly tampering with the connection. Quiting.")
659         }
660
661         if int(*k1) > len(alice.Sifted())/NMINPASS {
662                 log.Exit("Pick a better value for *k1")
663         }
664
665         log.Stderr("Starting reconciliation")
666         ch := make(chan int)
667         chBINARY := make(chan int)
668         chPass := make(chan Pass)
669         chParity := make(chan bool)
670         aliceCascade := Cascade{name: "Alice:", d: alice.Sifted(), ch: ch,
671                 chBINARY: chBINARY, chPass: chPass, chParity: chParity}
672         bobCascade := Cascade{name: "Bob   :", d: bob.Sifted(),
673         blocksize: int(*k1),ch: ch, chBINARY: chBINARY, chPass: chPass,
674         chParity: chParity}
675
676         aliceCascade.Init()
677         bobCascade.Init()
```

```
678
679         go aliceCascade.Alice()
680         bobCascade.Bob()
681
682         t := aliceCascade.Divulged()
683         log.Stderr("Divulged:", t)
684
685         tch := make(chan []bool)
686         aliceToeplitz := Toeplitz{data: aliceCascade.Data(),
687                 t: t, s: int(*s), ch: tch}
688         bobToeplitz := Toeplitz{data: bobCascade.Data(),
689                 t: t, s: int(*s), ch: tch}
690
691         go bobToeplitz.Bob()
692         aliceToeplitz.Alice()
693
694         log.Stderr(len(aliceToeplitz.data), "bits final key:")
695         log.Stderr("Alice:", BoolArray2IntArray(aliceToeplitz.data))
696         log.Stderr("Bob  :", BoolArray2IntArray(bobToeplitz.data))
697
698 }
```

bb84.go

BIBLIOGRAPHY

[1] G. Brassard, A Bibliography of Quantum Cryptography, SIGACT News 75 (1993)

[2] C.H. Bennett and G. Brassard Quantum Cryptography: Public Key Distribution and Coin Tossing, *Proceedings of IEEE International Conference on Computers Systems and Signal Processing*, 175-179 (1984)

[3] C. H. Bennett, Quantum Cryptography Using Any Two Nonorthogonal States, Phys. Rev. Lett. 68, 3121 (1992)

[4] A.K. Ekert, Quantum Cryptography Based on Bell's Theorem, Phys. Rev. Lett. 67, 661 (1991)

[5] C. H. Bennett, F. Bessette, L. Salvail, G. Brassard, J. Smolin, Experimental Quantum Cryptography, Journal of Cryptography, 5:3-28 (1992)

[6] G. Brassard, L. Salvail, Secret-key reconciliation by public discussion, Advances in Cryptology - Eurocrypt '93, Lecture Notes in Computer Science, 410-423 (1993)

[7] Einstein, A., Podolsky, B., Rosen, N., Can quantum-mechanical description of physical reality be considered complete?, Physical Review 47: 777-780. (1935)

[8] J.S. Bell, On the Einstein Podolsky Rosen paradox, Physics 1(3): 195-200 (1964)

[9] J.F. Clauser, M.A. Horne, A. Shimony, R.A. Holt, Phys. Rev. Lett. 23, 880 (1969)

[10] J. S. Bell, Speakable and Unspeakable in Quantum Mechanics (Cambridge University Press 1987)

[11] A. Aspect, P. Grangier, G. Roger, Phys. Rev. Lett. 49, 1804 (1982)

[12] B. Huttner and A. K. Ekert, Information gain in quantum eavesdropping, J. Mod. Opt. 41, 2455–2466 (1994)

[13] C.H. Bennett, G. Brassard, C. Crepeau, and U.M. Maurer, Generalized Privacy Amplification, IEEE Trans. Inform. Theory, vol. 41, 1915–1923 (1995)

[14] J.J. Sakurai, Modern Quantum Mechanics, Revised Edition, Addison-Wesley Publishing Company (1993)

[15] N.A. Chuang, I.L. Chuang, Quantum Computation and Quantum Information, Cambridge University Press (2000)

[16] N. Gisin, G. Ribordy, W. Tittel, and H. Zbinden, "Quantum cryptography," Rev. Mod. Phys. 74, 145. (2002)

[17] W.P. Schleich, Quantum Optics in Phase Space, Wiley (2001)

[18] R.P. Feynman, Feynman Lectures on Computation, Perseus Books (1996)

[19] V. Scarani, A. Acin, G. Ribordy, N. Gisin, Phys. Rev. Lett. 92, 057901 (2004)

[20] H. Imai, M. Hayashi, Quantum Computation and Information, Springer (2006)

[21] G. Assche, Quantum Cryptography and Secret-Key Distillation, Cambridge University Press (2006)

[22] C. Elliott, A. Colvin, D. Pearson, O. Pikalo, J. Schlafer, H. Yeh, Current status of the DARPA Quantum Network, arXiv:quant-ph/0503058v2 (2005)

[23] T. Sugimoto and K. Yamazaki. A study on secret key reconciliation protocol "cascade". IEICE Trans. Fundamentals, E83A No. 10:1987 (2000)