Quantum Key Distribution Protocol Literature Review and BB84 Simulation

Guthrie Cordone and Cameron Fabbri

CS456 Final Project

12/9/2014

1. Summary

We investigated Quantum Key Distribution methods, examined possible ways to hack these protocols, and developed a BB84 simulation code in Java. The quantum key distribution methods that we investigated were at the BB84 distribution method and the B92 distribution method. The quantum hacking methods investigated were a Trojan-horse attack and a photon number splitting attack. Using the knowledge from our investigations, we successfully developed a BB84 simulation program in Java that provides example functionality of the cryptosystem without needing real quantum circuitry.

2. Introduction

Throughout history, secret key cryptographic methods have been used during wartime to conceal communications between a commander and their troops. Examples of old cryptographic schemes used for war include the Caesar cipher during Roman times and the Enigma cipher during World War 2. As the computer age blossomed and more people began sending sensitive information over the internet, many saw that it was necessary to encrypt their information to prevent outsiders from seeing it. At this time, the use of secret keys moved from a primarily military use to a more public use. As a response to internet security concerns, mathematicians and computer scientists devised cryptographic systems for use over networks using number theory and mathematics. Two of the most common cryptographic key distribution systems developed during this time were RSA and El Gamal.

Classic cryptographic key distribution systems such as RSA and El Gamal depend on complex, convoluted mathematical problems to be successful. Specifically, the security of the RSA cryptosystem depends on the difficulty of factoring large numbers [1], while the security of El Gamal cryptosystem depends on the problem of computing the discrete logarithm over a finite field [2]. The inherent security flaw in using these problems for cryptography is that the key length needs to keep up with computer processing speed. In 1994, Peter Shor published a paper that described two quantum algorithms that could respectively solve the factoring problem and discrete logarithm problems [3]. Shor's landmark paper effectively doomed all implementations of RSA, El Gamal, and other classic cryptographic systems that used factoring or the discrete logarithm. Once a quantum computer is built, these systems will be easily broken.

There are some cryptographic systems that will still function beyond the invention of the quantum computer, despite Shor's doomsday algorithms. Two of these cryptosystems are the quantum key distribution methods BB84 and B92.

3. Quantum Key Distribution Methods

Quantum key distribution was first proposed in a landmark paper by Charles Bennet and Gilles Brassard in 1984 [5]. The idea behind quantum key distribution was inherently different than classical key distribution methods such as RSA and El Gamal. Instead of convoluted mathematical problems, quantum key exchange relies on the quantum properties of a particle, typically a photon. The quantum properites of particles do not agree with our basic ideas of physics, much to the dismay of Einstein, Podolsky, and Rosen as described in [6]. A list of the quantum properties used in quantum key distribution methods is provided in Gisin et al's quantum cryptography review in [7]. The described properties from [7] are listed in Table 1.

Table 1: Properties of Quantum Particles useful for Quantum Key Exchange Cryptosystems [7]

Property #	**Description**
Property 1	"One cannot take a measurement without perturbing the system."
Property 2	"One cannot determine simultaneously the position and the momentum of a particle with arbitrarily high accuracy."
Property 3	"One cannot simultaneously measure the polarization of a photon in the vertical-horizontal basis and simultaneously in the diagonal basis."
Property 4	"One cannot draw pictures of individual quantum processes."
Property 5	"One cannot duplicate an unknown quantum state."

All five of the properties described in Table 1 are particularly useful for quantum key exchange, and are the major security points in both the BB84 and B92 quantum key distribution methods.

3.1. BB84

Proposed in 1984 by Bennett and Brassard [5], BB84 was the first quantum key distribution method. BB84 used the Heisenberg Uncertainty Principle to securely send data between two clients.

In quantum mechanics, the internal state of a photon is represented by $|\psi\rangle$, which a vertical unit length vector over a field of complex numbers. When measured, the quantum particle will collapse to a state determined by the coefficients within $|\psi\rangle$. We can influence the coefficients within $|\psi\rangle$ by passing the photon through a polarization filter, which forces the spin of the photon into a certain angle [7]. Bennett and Brassard use two polarization filters, a rectilinear filter and a diagonal filter, to filter photons in BB84. Each filter interprets $|\psi\rangle$ with a different set of orthogonal basis: 90° for the rectilinear filter and 45° for the diagonal filter. The nature of the two filters is that if a photon with a spin in one basis is measured with a filter in the other basis, the output will have a random spin in the measured basis. If the photon is measured with a filter in the same basis, then the output will simply be the spin of the photon. The nature of these filters, along with all five properties of Table 1, are the crux of BB84.

In BB84, correspondents Alice and Bob need to share both a classical channel and a quantum channel. The classical channel is a typical electrical wire and the quantum channel is a fiber optic cable. They use the rectilinear and diagonal polarization filters to send and measure polarized photons (qubits) to each other over the quantum channel. They use the spin angle of the polarization to interpret as a bit value ove zero or one. Table 2 shows the encoding scheme for respective polarization angles. They then use the properties of the qubits to safely create a secret key and discuss over the classical channel. The step-by-step description of BB84 is listed in Table 3, using Alice and Bob as example correspondents.

Table 2: Photon Polarization Encoding Scheme for BB84

Basis	**Polarization Angle**	**Bit Value**
Rectilinear ("+")	0°	0
	90°	1
Diagonal ("X")	45°	0
	135°	1

Table 3: The BB84 Quantum Key Distribution Protocol

Step #	**Description**
Step 1	Alice chooses a random bit string and a random string of polarization bases.
Step 2	Alice sends Bob a string of photons, each with a polarization in the respective basis corresponding to the respective bit value of her bit string (see Table 2 for bit interpretations).
Step 3	Bob receives Alice's photons and individually measures each one with a random basis, acquiring the correct polarization if he measures using the same basis as Alice or a random polarization if he measures using a different basis than Alice. He interprets each polarization as a bit using Table 2.
Step 4	Bob announces to Alice that he has completed measuring her photons. Bob and Alice exchange basis strings.
Step 5	Bob and Alice compare basis strings, eliminating respective bits in their bit strings if they measured the photon with a different basis. The leftover bits should be identical to Alice and Bob.
Step 6	The bits corresponding to correctly measured photons are used as a secret key.

The security of the BB84 protocol lies in properties 3 and 5 of Table 1. Property 3 prevents an eavesdropper, Eve, from duplicating the photons sent to Bob. If Eve could duplicate the photons, she would simply have to wait for Bob and Alice to exchange bases over the public channel. With the bases and photons, she could easily generate their key using Step 5 in Table 3. Property 5 prevents a photon from being measured with two different bases at once, which would completely destroy the system if true, as anyone could determine the polarization of Alice's photons.

Eve could also intercept the photons, measure them, and generate her own random polarization photons to send to Bob. If Alice and Bob want to check for such an attack, they simply share a number of their bits generated during Step 5. If all of the bits are the same, then they assume that no eavesdropping took place. If at least one of the bits are different, then they know that someone interfered since all bits should be the same.

3.2 B92

In 1992, Bennett published a paper which simplified the BB84 protocol [8]. The main difference between B92 and BB84 was that B92 only used two nonorthogonal polarization states for binary encoding instead of four. The new polarization encoding for B92 is given in Table 4.

Table 4: Photon Polarization Encoding Scheme for BB92

Basis	**Polarization Angle**	**Bit Value**
Rectilinear ("+")	0°	0
Diagonal ("X")	45°	1

We see from Table 4 that B92 doesn't have a 0 and 1 bit code for each polarization basis, but instead uses the basis as the bit value. The measurement functions used in B92 erase the spin of the photon when measured in the wrong basis instead of randomly assigning it a polarization in the other basis. Therefore Bob knows which of Alice's photons he correctly measured. Other than using two states instead of four, the process for B92 is exactly the same as the BB84 process described in Table 3. In the end, the key size generated by B92 approaches 25% of the length of the vector Alice sends, while the key size generated by BB84 approaches 50%.

4. Quantum Key Distribution Attacks

In [11], Shor proved that the BB84 quantum key distribution protocol is completely unbreakable when using perfect photonic circuitry. The issue is that nowadays we are far from manufacturing perfect circuitry. Methods to hack quantum key distribution protocols rely on these imperfections of modern-day photonic circuitry.

In [9], Lydersen described the known methods of attack on quantum key distribution systems. These methods include the photon-number splitting attack and the Trojan-horse attack.

4.1. Photon-Number Splitting Attack

The photon-number splitting (PNS) attack relies on the infidelity of Alice's photon source. In the attack, Eve measures how many photons Alice sends to Bob per pulse. Measuring the number of photons does not change the polarization of them. If Alice sends more than one photon per pulse, Eve can store one for herself, and then measure it with Alices basis once announced, effectively generating the secret key.

One way to prevent a PNS attack is to use the decoy states protocol, which is a variant of BB84. In the decoy states protocol, Alice varies the number of photons per pulse and determines the fraction of detections by Bob, effectively determining if Eve intercepted any photons.

4.2. Trojan-Horse Attack

Another attack that uses the imperfection of the photonic circuitry is the Trojan-horse attack. In the Trojan-horse attack, Eve pulses an intense laser at Alice's or Bob's system between their photon transmissions. The reflections of the laser off of the target's system reveal the setting of their phase modulator, which contains the basis and bit values of the most recent sent photon. Using these values, Eve could easily compile the secret key.

One way to prevent a Trojan-horse attack is to place a beam attenuator on Alice's output fiber. The attenuator would decrease the intensity of Eve's reflection two times, once going in and once going out, effectively destroying Eve's pulse. To make up for the attenuator, Alice now has to send a very bright pulse per bit during transmission to Bob, since most of the pulse is lost.

5. BB84 Simulation

To simulate the BB84 protocol, we used the Java programming language, and uses a GUI for user input and displaying information. The purpose of this application is to simulate the creation of qubits in the most realistic way possible, and to use them to create a secret shared key between Alice and Bob. More on how those qubits are created will be covered later in this section.

5.1. The Qubit Class

Using Java it is fairly simple to create a qubit object, however, to keep the integrity of the simulation certain precautions must be considered. The nature of the qubit that Alice sends to Bob restricts Bob from finding its basis or polarization, so those must be hidden, or in Java terms, private. Our default constructor, which creates a new "Qubit" object with a basis and polarization, is how we create the simulated qubit object. A sample from the code is given:

```
	private String polarization, basis;

	public Qubit(String polarization, String basis)
	{
		this.polarization = polarization;
		this.basis        = basis;
	}
```

Notice that the polarization and basis are set to private. This restricts anything (Bob) outside the Qubit class from accessing it directly.

To generate the basis for a qubit, we used the Java Random class, which uses a 48-bit seed and is modified using a linear congruential formula [10]. To represent the bases, we use "+" and "X". The generator generates a number between 0 and 1 exclusive, and chooses a basis based on the number. If the number was less than 0.5, then the basis is represented by a "+", otherwise it is represented by an "X". It should be noted that because the cutoff is at 0.5, the scale is tipped very slightly in one direction (less than or equal to 0.5). However, the random generator generates a decimal of about 17 or 18 decimal places long, so the chances of getting exactly 0.5 are very slim. Our generate basis method is given as:

```
	public String generateBasis()
	{
		double basis_num = qubit.randomGenerator();
		if (basis_num < 0.5) basis = "+";
		else basis = "X";
		return basis;
	}
```

The qubit's polarization is randomly generated by the chosen basis. Given the "+" basis, a horizontal or vertical polarization will be given, which are represented by "-" and "|" respectively. Given a "X" basis, a polarization of 135° or 45° will be given, which are represented by "/" and "\" respectively. Our generate polarization method is given as:

```
	public String generatePolarization(String basis)
	{
		plus  = "+ ";
		plus_ = "+";
		X     = "X ";
		X_    = "X";

		if (basis.equals(plus) || basis.equals(plus_))
		{
			double polarization_num = qubit.randomGenerator();
			if (polarization_num < 0.5)
			{
				polarization = "|";
			}
			else
			{
				polarization = "-";
			}
		}
		else if (basis.equals(X))
		{
			double polarization_num = qubit.randomGenerator();
			if (polarization_num < 0.5)
			{
				polarization = "/";
			}
			else
			{
				polarization = "\\";
			}
		}

		return polarization;
	}
```

5.2. Methods

An important part of the process is Bob's measurement of Alice's qubits. This is how Bob generates his respective bit, which is used in his secret key. Measuring Alice's qubit also destroys them, which allows Alice to send her basis to Bob, and also prevents Eve from stepping in. The measure qubit method operates by first measuring Alice's basis with Bob's basis. If Bob's random basis matched with Alice's, then the bit corresponding to the polarization of that basis is used for their secret key. If they do not match, then the measured bits are thrown away. Either way, after the bit is set the qubit is destroyed. The generate bit method returns the bit 1 if the polarization is either "|" or "/", and a zero if the polarization is "-" or "\". The measure qubit and generate bit methods are given as:

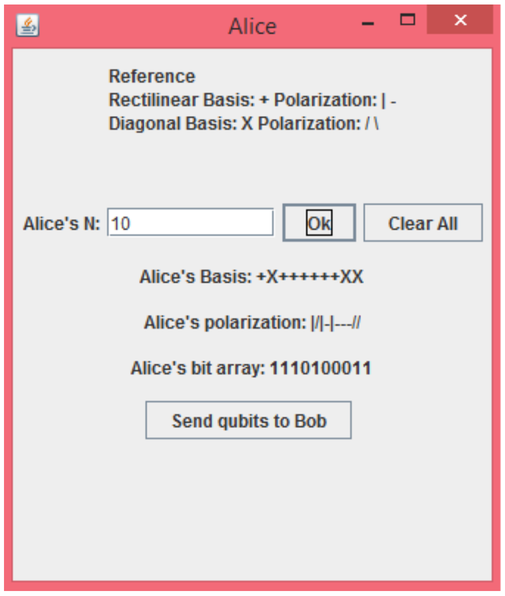
```
public int generateBit(String polarization)
	{
		if      (polarization.equals("|"))  return 1;
		else if (polarization.equals("/"))  return 1;
		else if (polarization.equals("-"))  return 0;
		else if (polarization.equals("\\")) return 0;
		else return -99;
	}

	public int measureQubit(Qubit myqubit, String basis_)
	{
		int bit;
		if (myqubit.basis.equals(basis_))
		{
			bit = generateBit(myqubit.polarization);
			myqubit = null;
		}
		else
		{
			myqubit.basis        = basis_;
			myqubit.polarization = myqubit.generatePolarization(basis_);

			bit = generateBit(myqubit.polarization);
			myqubit = null;
		}
		return bit;
	}
```

5.3. Using More Than One Qubit

Clearly, we won't be using just one qubit to generate a secret key. This section will explain the methods used to generate *N* qubits, and how they are used to generate the key.

Alice

Upon starting the simulation, Alice has a choice for an integer, N. N is the length of the array of qubits she wants to generate. Currently, the simulation handles qubit arrays of under 100,000 fairly well. Going to 100,000 and above will result in some lagging and will take considerably more time. After choosing an N, Alice will create an array of N qubits, and for informational purposes display her corresponding basis, polarization, and bit array. From there she can send her qubit array to Bob. The GUI for this step is given in Figure 1.



Reference
Rectilinear Basis: + Polarization: | -
Diagonal Basis: X Polarization: / \

Alice's N: 10

Alice's Basis: +X++++++XX

Alice's polarization: |/|-|---//

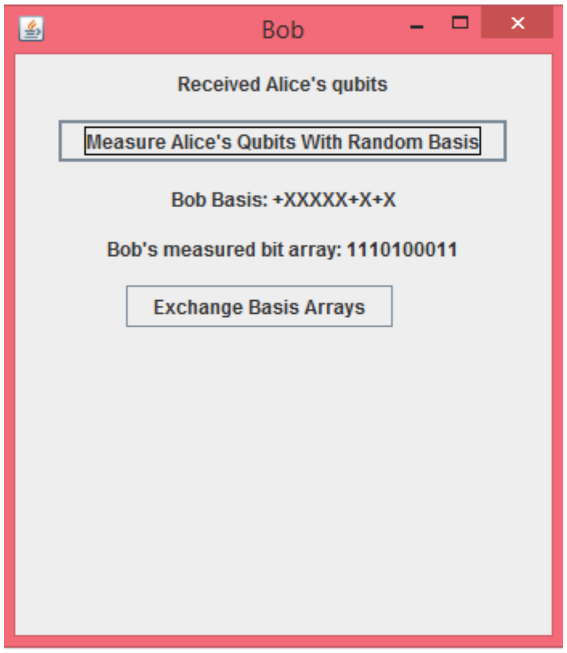
Alice's bit array: 1110100011

Figure 1: Alice picking N, generating a basis, polarization, and sending qubit array to Bob

Remembering our qubit constructor from earlier, to generate a qubit you must first have a basis and polarization. Alice first creates an array of bases by continually calling the generate basis method, and then uses that array to generate her polarization array in a similar fashion. Alice then uses those two arrays to create the resulting array of qubits to send to Bob. Because this is all happening within the simulation on a local machine, Alice doesn't technically "send" anything. The sending is simply a notification to Bob that Alice has created her qubit array of size N, and that she is ready for him to proceed with the next steps.

<u>Bob</u>

After "receiving" Alice's qubit array, Bob uses the N provided by the length to create his own basis array in the same fashion that Alice did. He then creates his own bit array by using the measure qubit method. Remember that by doing this, Alice's qubit array is destroyed. Again for information purposes, Bob's basis and bit array are displayed. The GUI for this step is given in Figure 2.



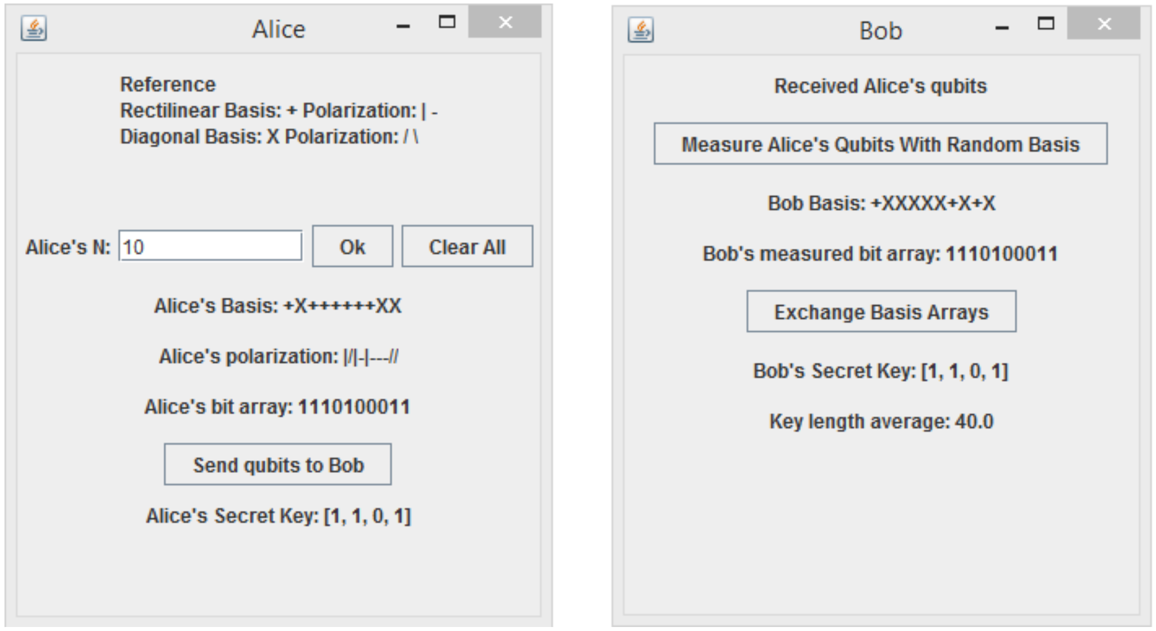
Received Alice's qubits

Bob Basis: +XXXXX+X+X

Bob's measured bit array: 1110100011

Figure 2: Bob measuring Alice's quibit array with his random basis array

Now that both Alice and Bob have a basis and bit array, they can exchange their basis arrays to parse their bit arrays. If the bases are the same, then the corresponding bit is kept. Otherwise, it's discarded. The result is a shared secret key, whose length on average is N/2. To analyze this, we keep track of the average length of the key and compare it to N. On each run, the percentage changes, and gradually converges to around 50%. Figure 3 shows the result of exchanging basis arrays. It can be seen that they have the same secret key.



Reference
Rectilinear Basis: + Polarization: | -
Diagonal Basis: X Polarization: / \

Alice's N: 10

Alice's Basis: +X++++++XX

Alice's polarization: |/|-|---//

Alice's bit array: 1110100011

Alice's Secret Key: [1, 1, 0, 1]

Received Alice's qubits

Bob Basis: +XXXXX+X+X

Bob's measured bit array: 1110100011

Bob's Secret Key: [1, 1, 0, 1]

Key length average: 40.0

Figure 3: Bob and Alice exchange basis arrays and generate a secret key

6. Analysis and Conclusion

Our BB84 simulator is far from complete. Although it serves well enough to show the principle and how it works, there is a lot that can be added. The simplest is to add in the method of actually sending the information between two computers over ports, and not just a simulated method of communication. The reason it was left out in this version was due to time constraints and level of importance with the overall goal of the simulation.

Adding in Eve would also be beneficial for our simulation. With Eve implemented, we would be able to test different security metrics of the protocol, such as how many bits on average Alice and Bob would need to share to detect Eve. From there we could test the other numerous modes of attack, and compare the security of this protocol with others either currently in use, or that have been used in the past.

Despite not having Eve and lacking true port communication, our simulation still demonstrates the basic idea and functionality of the BB84 quantum key distribution protocol.

References

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems Public-Key Cryptosystems."

[2] T. El Gamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," in *Proceedings of CRYPTO 84 on Advances in cryptology*, 1985, pp. 469 – 472.

[3] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," *Found. Comput. Sci. 1994 Proceedings., 35th Annu. Symp.*, pp. 124–134, 1994.

[4] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," pp. 124–134, Aug. 1995.

[5] C. Bennett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing," *Proc. IEEE Int. Conf. Comput. Syst. Signal Process.*, vol. 1, pp. 175–179, 1984.

[6] A. Einstein, B. Podolky, and N. Rosen, *Can Classical Description of Physical Reality Be Considered Complete?*. 2009.

[7] N. Gisin, G. Ribordy, W. Tittel, and H. Zbinden, "Quantum cryptography," *Rev. Mod. Phys.*, vol. 74, no. January, pp. 145–195, 2002.

[8] C. Bennett, "Quantum cryptography using any two nonorthogonal states," *Phys. Rev. Lett.*, vol. 68, no. 21, pp. 3121–3124, 1992.

[9] L. Lydersen, "Practical security of quantum cryptography," 2011.

[10] Java Random Class: http://docs.oracle.com/javase/7/docs/api/java/util/Random.html

[11] P. W. Shor and J. Preskill, "Simple proof of security of the BB84 quantum key distribution protocol," *Phys. Rev. Lett.*, vol. 85, pp. 441–444, 2000.