Jurnal Teknologi

Full paper

An Efficient Reconciliation in Removing Errors Using Bose, Chaudhuri, Hocquenghem Code for Quantum Key Distribution

Riaz Ahmad Qamar[a*], Mohd Aizaini Maarof[a], Subariah Ibrahim[a]

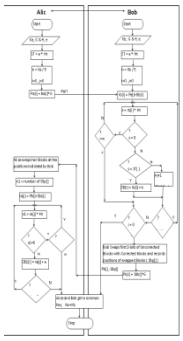
[a]*Faculty of Computer Science and Information Systems, Universiti Teknologi Malaysia, Johor Bahru, Johor Malaysia*

*Corresponding author: rzaqmr@gmail.com

Article history

Received : 10 February 2012
Received in revised form : 31 May 2012
Accepted : 13 August 2012

Graphical abstract



Abstract

A quantum key distribution protocol(QKD), known as BB84, was developed in 1984 by Charles Bennett and Gilles Brassard. The protocol works in two phases which are quantum state transmission and conventional post processing. In the first phase of BB84, raw key elements are distributed between two legitimate users by sending encoded photons through quantum channel whilst in the second phase, a common secret-key is obtained from correlated raw key elements by exchanging messages through a public channel e.g.; network or internet. The secret-key so obtained is used for cryptography purpose. Reconciliation is a compulsory part of post processing and hence of quantum key distribution protocol. The performance of a reconciliation protocol depends on the generation rate of common secret-key, number of bits disclosed and the error probability in common secrete-key. These characteristics of a protocol can be achieved by using a less interactive reconciliation protocol which can handle a higher initial quantum bit error rate (QBER). In this paper, we use a simple Bose, Chaudhuri, Hocquenghem (BCH) error correction algorithm with simplified syndrome table to achieve an efficient reconciliation protocol which can handle a higher quantum bit error rate and outputs a common key with zero error probability. The proposed protocol efficient in removing errors such that it can remove all errors even if QBER is 60%. Assuming the post processing channel is an authenticated binary symmetric channel (BSC).

Keywords: BB84; Reconciliation protocol; common secret-key; quantum bit error rate; simplified syndrome table; zero error probability

1.0 INTRODUCTION

The well-known conventional ciphers e.g.; RSA, Diffie-Hellman and AES are based on the computational difficulty techniques. Due to increase in computational power, powerful techniques have been developed to encrypt information/data but at the same time hackers can utilize this high computational power to decrypt it. These well-known ciphers lacks in providing secrecy proof and in detecting eavesdroppers. For example RSA algorithm, which is broadly utilized for key distribution[1], depends upon the unverified computational presumptions. If someone devises a faster technique for factoring large integers then the amount of computation time considerably reduces to decrypt information. Thus in the presence of high computational power, a key might be broken easily or an attacker can apply brute-force to decrypt short-key encrypted messages. Furthermore, Peter W. Shor developed an algorithm in 1994 which can be run on a quantum computer to reverse a one-way function[2]. Among other things, Kirchhoff`s principle states that security of a cryptosystem depends only upon the secrecy of the key[3] but not on the secrecy of the algorithm. A key is an important component of cryptography and the major problem in conventional cryptography is distribution of key among legitimate users. Thus quantum cryptography, alternatively named as quantum key distribution (QKD), provides a secure way of key distribution up to an acceptable level. Then, quantum-distributed key is used with secret-key cipher or with the prefect secure cipher, also known as one-time pad, for secure communications. The impregnability of both algorithms, e.g.; key distribution and encryption, is equally important otherwise whole security is compromised.

Quantum cryptography[4,5] is an emerging form of encryption which uses the laws of quantum physics, e.g.; the principle of photon polarization and Heisenberg uncertainty principle, to encrypt information on physical level. The principle of photon polarization describes polarization of light photons in a particular direction whereas; the Heisenberg uncertainty principle states that measurement of quantum state of a system is not possible without disturbing that system[6,7]. Because of the following principles the above laws make quantum cryptography secure:

(i) A quantum photon cannot be split to make its measurements secretly.

(ii) It is impossible to copy or clone a single photon to measure its state.
(iii) If an eavesdropper tries to measure directly the state of a photon, he will produce errors and can be detected.

In quantum key distribution (QKD) protocol, BB84, a key to be used for cryptography is distributed between two legitimate parties conventionally called Alice and Bob. The BB84 protocol works in two stages. In the first stage, known as quantum state transmission, a raw key is transmitted through a quantum channel using polarized/entangled photons or laser pulses as information carriers. The transmitted raw key may have errors due to the malfunctioning of the equipment used in quantum channel or due to an adversary who tries to discover the key. In the second stage, the raw key is distilled[8] by passing through processes of sifting, error estimation, reconciliation and privacy amplification. The key distillation is carried out by exchanging messages through public channel between Alice and Bob to obtain a common-secret key. The reconciliation process removes the errors in the shared key to have a common key between Alice and Bob, whilst privacy amplification compresses the common key to produce a secret-key. A reconciliation protocol is said to be (i) efficient in removing errors if it performs at high bit error rates and produces a common key with minimum error rate and (ii) efficient in terms of disclosed bits if it is less interactive[9]. A less interactive protocol corrects the errors with minimum iterations between Alice and Bob to get higher secret-key generation rate. An efficient reconciliation protocol, in terms of (i) and (ii) above, is required in QKD protocol.

Error correction method is an essential part of a reconciliation protocol. A binary BCH (Bose *et al*) code is one of the best options to correct errors in the raw key. Encoding in BCH code is a simple but decoding is a complex phenomenon in this code. A BCH code may produce wrong results when errors, produced in a transmitted codeword, are more than its error correction capability. For example, PGZ, Euclidean and Berlekamp-Massaey decoders[10] cannot provide solutions for this problem. These decoders may introduce additional errors when errors exceed their correction capability. The same may happen while using error patterns and their corresponding syndrome patterns ($e \rightarrow S$) lookup table for decoding of BCH codes[11].

The remaining part of the paper is arranged as follows: Section 2 explains popular reconciliation protocols. Section 3 describes the background of BCH error correction codes. Section 4 explains the proposed key reconciliation protocol. Section 5 describes the implementation of the protocol and Section 6 discusses simulation and results obtained from the protocol simulation. In the last section 7 closing remarks are given.

2.0 POPULAR RECONCILIATION PROTOCOLS

A reconciliation protocol is an important part of quantum key distribution protocol. Reconciliation protocols can process discrete or continuous random variables. They can work interactively or one way. Discrete random variables may be binary (0,1) or non-binary (alphabets). However, a key having continuous-variables is first converted into binary symbols before it reconciled. Generally, an error reconciliation protocol is said to be an ideal protocol which corrects all errors without introducing additional errors in the key and exhibits minimal information on the key to an adversary during reconciliation process. A reconciliation protocol permutes bits of key randomly and groups these bits into blocks. Then it computes parities of the blocks and exchanges these parities with its counterpart at the other end for comparison. If the parity of a block matches, the block is considered correct otherwise the erroneous bit is searched and corrected. The protocol repeats the process with different permutations and blocks sizes and continues until no disagreement is found in many subsequent comparisons of blocks parities. There are many types of error reconciliation protocols but more famous binary reconciliation protocols, working on the above stated principle, are BBBSS (Bennett-Bessette-Brassard-Salvail-Smolin)[12], Cascade[13,14,15], Farkawa-Yamazaki (FY)[16], and Winnow[9,17,18].

2.1 Bennett-Bessette-Brassard-Salvail-Smolin Protocol

Bennett-Bessette-Brassard-Salvail-Smolin Protocol (BBBSS) is a first binary interactive error correction (IEC) protocol used for key reconciliation in quantum key distribution. It is designed by Bennett and his coworkers. This protocol can handle a long binary string. Alice and Bob exchange parities of their blocks and compare these parities. The presences of diverging parities (parities that are not equal) help Alice and Bob to focus on errors using bisection and to correct those. It uses several iterations, between which the bit positions are permuted in a pseudo-random way.

2.2 Cascade Protocol

This reconciliation protocol is based on BBBSS protocol but with an improved efficiency in term of the number of disclosed bits. It uses four(4) iterations. First iteration is identical to the first iteration of BBBSS, while the next three are different. Unlike BBBSS, it records the result of all previously investigated blocks and takes advantage of this information in next iteration. It keeps two sets of blocks: (i) the blocks for which the parity is equal between Alice and Bob (ii) the blocks of diverging parity. Each block for which parity was disclosed, during the bisection, is listed either (i) or (ii). It optimizes the block size in turn reduces the number of bits disclosed. It optimizes bit interleaving between two iterations.

2.3 Furukawa – Yamazaki Protocol

This IEC protocol based on BBBSS and uses perfect code designed by Furukawa and Yamazaki. Like BBBSS, it uses a certain number of iterations with bit interleaving in between. It cuts the string into blocks like BBBSS and thus determines which block contains an odd number of errors. Instead of using interactive bisection, the correction of erroneous blocks is one way from Alice to Bob. For each block with a diverging parity, Alice sends Bob the syndrome of a perfect code calculated over her block. With this information, Bob corrects his block. This protocol is less efficient than cascade in terms of the number of bits discarded.

2.4 Winnow Protocol

Winnow protocol is very similar to FY protocol. It also includes a privacy maintenance step that discards bits during the error correction. It uses a certain number of iterations with bit interleaving in between. Like BBBSS, FY and Cascade, it also cuts binary key into blocks. Alice and Bob exchange parities of their blocks and thus determine which blocks contain an odd number of errors. For blocks of diverging parity, Alice sends Bob the syndrome of a Hamming code calculated over her block. Unlike BBBSSS and Cascade, which use a bisection, the correction of a block using the Hamming code does not necessary

reduce the number of errors in that block. The hamming code proposed in winnow allows Alice and Bob to correct one error. If more than one error present in the block, Bob's attempt may actually increase the number of errors in that block. Thus, block size should be chosen in such a way that it globally reduces the number of errors. Unlike Cascade, the iterations of Winnow are independent of each other and so an exhaustive search could be performed at a low complexity using dynamic programming. Winnow protocol performs efficiently in terms of disclosed bits between 10% and 18% quantum bit error rate(QBER). Winnow can achieve about 10^{-3} final key bit error rate by optimizing block size. While CASCADE protocol is preferred between 18% and 25% QBER[19]. Other suggested binary reconciliation protocols, where authors use BCH[20] and LDPC[21] error correction methods, can achieve 10^{-5} and 10^{-6} final key bit error rate respectively.

3.0 BACKGROUND OF BCH CODES

An error correcting code consists of techniques and algorithms and has two fundamental operations: encoding and decoding. Encoding is a procedure in which redundancy (parity) is added to a message and transforms it into a code word. This code word is transmitted through a noisy channel, which changes the message. Decoding procedure removes the errors from erroneous code word and gets the message back into its original form. The block codes process the information block-by-block and treat each block independently. In linear block codes, adding two code words produces another code word. A linear code involves generator and parity-check matrices, **G** & **H** respectively. Cyclic codes are codes in which cyclically shifting the symbols of a code word produces another code word. In binary codes, data is processed as binary digits (0 or 1).

A binary BCH (Bose *et al*) code is a cyclic linear block code defined over Galois Field(2)[22]. Its error correcting capacity depends upon the minimum Hamming distance, d_{min}, between its code words. For example BCH code with $d_{min} = 3$ can correct any single error pattern, with $d_{min} = 5$ can correct double error pattern while with $d_{min} = 7$ can correct triple error pattern. The no-error case corresponds to the all-zero pattern. In general, a binary BCH code with code length $n = 2^m-1$ and minimum Hamming distance $d_{min} \geq 2t+1$ can correct any error pattern of size t or less where m is a positive integer and $m \geq 3$. In other words, this code can be designed to correct any error pattern of size t in a given code vector of length n where $n = 2^m-1$. The designed method of BCH code is based on the lowest common multiple (LCM) of a minimal polynomial.

In a linear cyclic block code, C(n,k), a code word also called code vector $\mathbf{c} = (c_0, c_1, c_2, \ldots c_{n-1})$ is transformed into another code word of this code after i times cyclic right-shift rotation as $c^{(i)} = (c_{n-i}, c_{n-i+1}, \ldots c_{n-1}, c_0, c_1, c_2, \ldots c_{n-i-1})$. The code vector form can be presented into a code polynomial form as $c(x) = c_0 + c_1x + c_2x^2 + \ldots + c_{n-1}x^{n-1}$. In a linear cyclic block code there is always a non-zero minimum-degree polynomial which is known as generator polynomial g(x), and any other polynomial of this code is a multiple of g(x). The degree of the generator polynomial is at most *mt* whereas any of other polynomials of the code has degree less than or equal to 'n-1'. Thus, a message vector m(x) can be encoded in this code as c(x) = m(x)g(x). A BCH(15,5,7) code with code length n =15, message length k = 5 and minimum Hamming distanc $d_{min} = 7$, has a generator polynomial $g(x) = 1+x^2+x^5+x^6+x^8+x^9+x^{10}$[10]. This generator polynomial can be expressed into a code generator matrix **G** of order k × n. After performing some row operations (Gaussian elimination), the generator matrix **G** is transformed into systematic form such that $\mathbf{G} = [\mathbf{P}_{k\times n-k} \ \mathbf{I}_k]$ as under:

$$\mathbf{G} = \begin{bmatrix} 1&0&1&0&0&1&1&0&1&1&1&0&0&0&0 \\ 1&1&1&1&0&1&0&1&1&0&0&1&0&0&0 \\ 0&1&1&1&1&0&1&0&1&1&0&0&1&0&0 \\ 1&0&0&1&1&0&1&1&1&0&0&0&0&1&0 \\ 0&1&0&0&1&1&0&1&1&1&0&0&0&0&1 \end{bmatrix}$$

To encode a message vector m(x), this matrix **G** of order (5×15) is multiplied with the message vector of order (1×5) to obtain a code vector i.e.; $\mathbf{c} = \mathbf{m} \times \mathbf{G}$. The parity check matrix **H**, which is used in decoding process, can be found from **G** as $\mathbf{H} = [\mathbf{I}_{n-k} \ \ \mathbf{P}^T_{n-k\times k}]$, where $\mathbf{P}^T$ is the transpose of matrix **P**[22,23]. So the corresponding parity check matrix **H** is formed as:

$$\mathbf{H} = \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&1&1&0&1&0 \\ 0&1&0&0&0&0&0&0&0&0&0&1&1&0&1 \\ 0&0&1&0&0&0&0&0&0&0&1&1&1&0&0 \\ 0&0&0&1&0&0&0&0&0&0&0&1&1&1&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0&1&1&1 \\ 0&0&0&0&0&1&0&0&0&0&1&1&0&0&1 \\ 0&0&0&0&0&0&1&0&0&0&1&0&1&1&0 \\ 0&0&0&0&0&0&0&1&0&0&0&1&0&1&1 \\ 0&0&0&0&0&0&0&0&1&0&1&1&1&1&1 \\ 0&0&0&0&0&0&0&0&0&1&1&0&1&0&1 \end{bmatrix}$$

If **c** is a transmitted vector, $\mathbf{r} = (r_0, r_1, r_2, \ldots r_{n-1})$ is a received vector and $\mathbf{e} = (e_0, e_1, e_2, \ldots e_{n-1})$ is an error pattern introduced by transmission noisy-channel then $\mathbf{r} = \mathbf{c} + \mathbf{e}$. In the decoding process, first of all, a syndrome vector $\mathbf{s} = (s_0, s_1, s_2, \ldots s_{n-k-1})$ is calculated with $\mathbf{s} = \mathbf{rH}^T$, where $\mathbf{H}^T$ is transpose of **H**. It means syndrome $\mathbf{s} = (\mathbf{c} + \mathbf{e})\mathbf{H}^T$ or $\mathbf{s} = \mathbf{cH}^T + \mathbf{eH}^T$. Now if transmitted code word has no error, as $\mathbf{e} = \mathbf{0}$, then $\mathbf{s} = \mathbf{cH}^T = \mathbf{0}$, otherwise $\mathbf{s} = \mathbf{rH}^T \neq \mathbf{0}$. Finally, decoding is implemented by using a syndrome table which is built to show the relationship between error patterns and their corresponding syndrome patterns ($\mathbf{e} \rightarrow \mathbf{s}$).

4.0 PROPOSED RECONCILIATION PROTOCOL

This protocol can be used to reconcile a raw key which is distributed through a quantum channel obeying quantum mechanics principles. The raw key contains errors because of malfunctioning of communication devices used and possible eavesdropping in a quantum channel. It is assumed that Alice and Bob hold a shared binary erroneous-key at the end of quantum state transmission process in QKD. A proposed protocol, which is used to reconcile a shared binary key, is shown in Figure 1 and is explained as follows:

(i) Alice and Bob generate a syndrome table ST[] from all possible error patterns in 5-bit message block as in Table 2.

(ii) Alice divides her key into blocks of 5-bit each and computes parity bits of their blocks as $\mathbf{p_a}[i] = \mathbf{B_a}[i] \times \mathbf{G}$.

(iii) Alice sends the computed parity bits $\mathbf{p_a}[i]$ to Bob through an authenticated public communication channel.

(iv) Bob divides his key into 5-bit blocks and combines parity bits, received from Alice, with these blocks at lower order positions respectively to form received vectors $\mathbf{r_b}[i]$ such as $\mathbf{r_b}[i] = (p_{a0}, p_{a1} \ldots\ldots p_{a9}, m_{b0}, m_{b1}, \ldots m_{b4})$

(v) Bob then calculates syndromes of vectors $\mathbf{r_b}[i]$ as $\mathbf{s}[i] = \mathbf{r_b}[i] * H^t$. Here there may be one of the two possibilities, either (a) calculated syndrome is equal to all-zero or (b) it is not all-zero vector. If syndrome is not equal to all-zero vector, then it may present in the syndrome table ST[] or not present in ST[].

(vi) Bob checks that if **s**[i] is all-zero then he leaves the block as it has no error. If **s**[i] is not all-zero and is present in the syndrome table then he adds error pattern taken from table ST[], corresponding to the computed syndrome pattern, to his block $\mathbf{B_b}[i]$ to get corrected the block as $\mathbf{CB_b}[i] = \mathbf{B_b}[i] + \mathbf{e}$. Now If **s**[i] is neither all-zero nor present in syndrome table then it means more than 3-errors exist in the block, it is beyond the capability of the BCH(15,5,7) code and Bob leaves it uncorrected and records the position of this uncorrected block as $\mathbf{UB_p}[j] = i$.

(vii) After processing all bocks, Bob see if there is any uncorrected block. If no uncorrected block left (i.e.; j = 0) it means all of errors in the key are corrected otherwise Bob swaps first two bits of uncorrected blocks with corrected or errorless blocks and records the positions of swapped blocks.

(viii) Bob calculates the parity bits of swapped blocks (swapped blocks = 2 * uncorrected blocks) as $\mathbf{p_b}[] = \mathbf{SB_b}[] \times \mathbf{G}$ and sends these parity bits of each swapped block to Alice with positions of the swapped blocks $SB_p[]$ in the key.

(ix) Alice swaps her key blocks at positions indicated by Bob and then combines parity bits sent by Bob with her swapped blocks to get $\mathbf{r_a}[j]$.

(x) Alice calculates syndromes of the swapped blocks as $\mathbf{s_1} = \mathbf{r_a}[j] \times \mathbf{H^t}$ and compare these syndromes with the syndromes in table ST[] and adds corresponding error patterns to the swapped blocks to correct errors.

(xi) Finally, Alice and Bob retain a common key i.e.; Alice's key is same as the key of Bob.

5.0 IMPLEMENTING THE PROTOCOL

The proposed protocol is implemented in C-sharp (C#) computer language using two separate programs one for each player Alice and Bob. In addition, a third program is written which (i) generates a random key for Alice say KeyA, (ii) adds errors to KeyA to generate another key for Bob say KeyB. Errors are added to each block of KeyA to form KeyB as specified in Table 1. Each key consists of one million binary bits (0 or 1) and the size of each block is taken as five binary bits. Table 1 indicates that different sets of KeyA and KeyB are taken for simulation purpose. For example in data set #1, every block of KeyA and KeyB differs at most three numbers of binary positions. In data set #2, half of blocks of both keys differ at 4 or more positions and remaining 50% blocks may have less or equal to 3 errors.

Each of the two programs ProgramA and ProgramB computes a generator matrix **G** and parity check matrix **H** of BCH(15,5,7) code which are already mentioned in section 3.0. The remaining part of each program works as instructed in proposed protocol in section 4.0 for each player. Decoding Table 2 shows the relationship between error patterns and their syndromes which is constructed by multiplying possible error patterns of code BCH(15,5,7) with transpose of the parity check matrix **H** such as $\mathbf{s} = \mathbf{eH^T}$. The total number of error patterns of code BCH(n,k ,d_{min}), with error correcting capability equal to t

(t = $\frac{d_{min}-1}{2}$) , are calculated by $\sum_{i=0}^{t} C_i^n$

where C_i^n means $\frac{n!}{(n-i)! \times i!}$.

Hence total number of error patterns for BCH (15,5,7) code are 25.

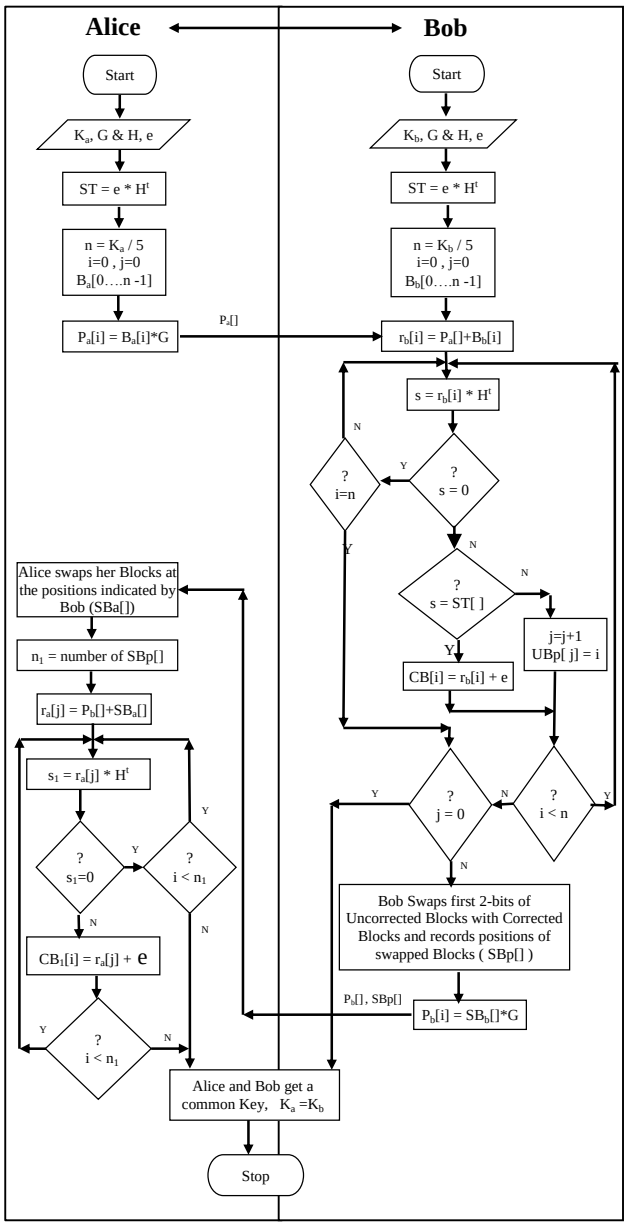


Figure 1 Flowchart of protocol for the reconciliation of key

Table 1 Error distribution in raw key

Data Set #	# of errors in each block, t	# of blocks in error (%)
1	≤ 3	100
2	≥ 4	50
3	≥ 4	55
4	≥ 4	60

Table 2 A simplified Syndrome Table ST[]

Syndromes	Error Patterns
0100110111	000000000000001
1001101110	000000000000010
1101011001	000000000000011
0111101011	000000000000100
0011011100	000000000000101
1110000101	000000000000110
1010110010	000000000000111
1111010110	000000000001000
1011100001	000000000001001
0110111000	000000000001010
0010001111	000000000001011
1000111101	000000000001100
1100001010	000000000001101
0001010011	000000000001110
1010011011	000000000010000
1110101100	000000000010001
0011110101	000000000010010
0111000010	000000000010011
1101110000	000000000010100
1001000111	000000000010101
0100011110	000000000010110
0101001101	000000000011000
0001111010	000000000011001
1100100011	000000000011010

Alice (ProgramA) divides the keyA into 5-bit message blocks say $\mathbf{B_a}$'s and computes their parity bits as $\mathbf{B_aG}$. The number of parity bits of each block is n-k=10 which are stored in a file say EncodedA. The file EncodedA is sent to Bob (ProgramB). Bob divides the KeyB into 5-bit blocks and attaches these blocks with respective parity bits taken from EncodedA file to make received words $\mathbf{r_b}$. Now each received word having 15 binary bits is multiplied by transpose of the parity check matrix **H** to calculate syndrome as $\mathbf{s} = \mathbf{r_aH}$. If **s** is a zero vector then no error exists in the Bob's block otherwise either error(s) exists in block or errors exceed the error correcting limit that is 3 of the BCH(15,5,7) code. When calculated syndrome **s** is a non-zero vector then it is searched in the decoding Table 2. If **s** is found in the table, the corresponding error pattern is XORed with Bob's block to correct error(s) in the block otherwise block marked as uncorrected block. This uncorrected block has more than 3 errors which are beyond the error correcting limit of the code. At this stage one pass is completed. Now if error distribution in the Bob's key is such that each block contains errors less or equal to 3 then all the errors in the key are corrected in one pass. On the other hand, if uncorrected blocks are found in the key then Bob swaps first three bits of each uncorrected block with no error or corrected blocks to scatter errors with in blocks. In this way, swapped blocks, which are double in number of the uncorrected blocks, has at most three(3) errors as compared to Alice's key blocks at the corresponding positions. In case of uncorrected blocks found, Bob encode all of the swapped blocks and sends parity bits to Alice with the swapped block positions. Then Alice (ProgramA) decodes her corresponding blocks by using the information provided by Bob. This is the second pass and all the errors are removed in the key and thus a common-key is obtained at both ends.

6.0 RESULTS AND DISCUSSIONS

This protocol is implemented in C-sharp (C#) computer language as explained in section 5.0. A binary key, as KeyA, is generated randomly and errors are introduced in the generated key to form another key, as KeyB. Four sets of these keys, having different number of blocks in error, are prepared as shown in Table 1. Each key consists of one million binary bits (0 or 1). After processing these two keys for error correction, both keys are found to be equal. The experiment is repeated 100 times.

Table 3 reveals the working comparison between four popular protocols and the proposed protocol. The working information is taken from the literature as referred in section 2.0. All the four protocols correct single error in a block at a time while our protocol corrects upto three errors per block. Winnow protocol introduces an additional error instead of correcting when errors per block are more than 1 and parity of the block is odd. In proposed protocol 1 or 2 iterations (passes) are required to correct the key while Cascade protocol required 4 iterations and other three protocols required several iterations. Block size and bit-interleaving optimization is used in Cascade, winnow and prosed protocol whilst BBBSS and Yamazaki protocols do not require these optimizations. Also, syndrome decoding Table 2 is used to correct upto three errors and to detect more than three errors. The comparison discussion above indicates that the suggested protocol is efficient in removing errors.

Table 3 A working comparison of existing binary interactive reconciliation protocols with proposed protocol

Working-steps	BBBSS	Cascade	Furkawa-Yamazaki(FY)	Winnow	Proposed protocol
Divides the key into blocks	√	√	√	√	√
Both parties interchange their blocks parities	√	√	√	√	x
Uses binary-search for correcting errors	√	√	Uses perfect code to correct errors. Blocks syndromes are sent one-way, from Alice to Bob	Error corrected by Hamming code interactively	Error corrected by BCH code

Uses several iterations	√	Uses 4 iterations	√	√	Uses 1 or 2 iterations
Uses bit permutations between two iterations	√	√	√	√	Swapping bits of blocks
Keeps record of investigated blocks	x	√	x	x	Only uncorrected blocks
Optimizes block size between two iterations	x	√	x	√	x
Optimizes bit interleaving between two iterations	x	√	x	√	x

Second column of Table 4 shows number of errors introduced per block of size 5-bit in Alice's key (KeyA) to generate Bob's key (KeyB) and the third column indicates total number of blocks which have errors mentioned in column two while comparing keyA and KeyB. After implementing the proposed protocol using these keys as input, a common key is achieved with Alice and Bob. The number of blocks corrected and the size of common key retained is shown in column four and five respectively.

Table 4 shows remaining key-size when different number of errors is present in different number of blocks. All blocks are corrected if every block has errors less or equal to 3 (data set 1) or at least half of the blocks contain less that 4 errors per block. But when per block error-limit exceeds than 3 (≥ 4) then retained key-size depends upon the number of blocks in error. As the number of erroneous blocks having errors greater than 3 increases the retained key-size decreases. Thus, if more than 50% blocks having errors greater or equal to 4 per block then extra erroneous blocks (>50%) are discarded during the first pass of protocol to obtain zero-error common key and hence the size of the achieved common key is reduced. The last two rows of the Table 4 predict this type of situation. However, the suggested reconciliation protocol can handle a key with 60% ($\frac{3}{5}\times 100$, data set # 1) initial bit error rate and can produce full-size common key with zero-error probability whereas winnow protocol performs efficiently at upto 18% initial bit error rate and Cascade at up to 25% and the common key obtained in Winnow, BCH and LDPC protocols with 10^{-3}, 10^{-5} and 10^{-6} final key bit error rate respectively

Table 4 Key correction with proposed reconciliation protocol using 5-bit block size

Data Set #	# of errors in each block, t	# of blocks in error (%)	# of blocks corrected (%)	Key-size retained (%)
1	≤ 3	100	100	100
2	≥ 4	50	100	100
3	≥ 4	55	90	90
4	≥ 4	60	80	80

7.0 CONCLUSION

The protocol, mentioned in this paper, is a fast and efficient protocol to correct errors in a key which is generated in quantum key distribution (QKD) scheme. This protocol has the capability to correct errors even if all blocks of a raw key are in error provided that each block contains errors equal to or less than 3. Blocks containing errors more than three can also be detected by the use of decoding table. It can work successfully for long keys. It is able to correct all the errors in the key with maximum of two passes (iterations between Alice and Bob) which supports key secrecy. Thus error probability in the retained key is zero. The suggested protocol only sets the limits on errors per block and number of blocks in error. Alice and Bob exchange more parity bits as compare to other famous protocols e.g.; BBBSS, Cascade and Winnow.

References

[1] Elboukhari M., A. Azizi, M. Azizi. 2009. Implementation of Secure Key Distribution Based on Quantum Cryptography. *International Conference on Multimedia Computing and Systems*. 361–365.

[2] Shor P. 1994. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. Proceedings of the 35th Annual Symposium on Foundations of Computer Science. Santa Fe, NM, IEEE Computer Society Press. 124–134.

[3] Assche G. 2006. *Quantum Cryptography and Secret-Key Distillation.* 6th. ed. New York: Cambridge University Press.

[4] Alléaume R. 2007. SECOQC White Paper on Quantum Key Distribution and Cryptography. *Secoqc-WP-v5, White Paper, Ver. 5.1*.

[5] Teja1 V., P. Banerjee, N. N. Sharma, R. K. Mittal. 2007. Quantum Cryptography: State-of-Art, Challenges and Future Perspectives. Proceedings of the 7th IEEE International Conference on Nanotechnology, Hong Kong. 1296–1301.

[6] Hrg D., L. Budin, M. Golub. 2004. Quantum Cryptography and Security of Information Systems. Proceedings of the 15th Conference on Information and Intelligent System, IEEE. 63–70.

[7] Papanikolaou N. 2005. An Introduction to Quantum Cryptography. *ACM Crossroads Magazine.* 11(3): 1–16.

[8] Van D., V. Tilborg. 1998. The Art of Distilling [Secret Key Generation]. *Information Theory Workshop, IEEE*. 158–159.

[9] Buttler W. T., S. K. Lamoreaux, J. R. Torgerson, G. H. Nickel, C. H. Donahue, and C. G. Peterson. 2003. Fast, Efficient Error Reconciliation for Quantum Cryptography.

[10] Robert H. Morelos-Zaragoza. *The Art of Error Correcting Coding*, 2nd. Ed. 2006. England. John Wiley & Sons.

[11] Lee H. P., H. C. Chang, T. C. Lin, and T. K. Truong. 2008. A Weight Method of Decoding the Binary BCH Code. IEEE Eighth International Conference on Intelligent Systems Design and Applications.

[12] Bennett C.H., F. Bessette, G. Brassard, L. Salvail, and J. Smolin. 1992. Experimental Quantum Cryptography. *Journal Cryptology*. 5(1): 3–28.

[13] Brassard G., and L. Salvail. 1993. Secret-key Reconciliation by Public Discussion. Advance in Cryptology–Eurocrypt '93. *Lecture Notes in Computer Science*. 410–423.

[14] Koichi Y., Ranjith N., and Horace P. Y. 2008. Problems of the CASCADE Protocol and Renyi Entropy Reduction in Classical and Quantum Key Generation. Available from e-print archive arXiv.org , record: quant-ph/0703012v1.

[15] Bellot P., Minh-Dung D. 2009. BB84 Implementation and Computer Reality. International Conference on Computing and Communication Technologies, 2009, IEEE.1–8.

[16] Furukawa E., and K. Yamazaki. 2001. Application of Existing Perfect Code to Secret Key Reconciliation. *Conf. Proc. Int. Symp. Commun. Inform*. Tech. 397–400.

[17] Feng Zhao, Mingxing Fu, Faqiang Wang, Yiqun Lu, Changjun Liao, Songhao Liu. 2007. Error Reconciliation for Practical Quantum Cryptography. *International Journal for Light and Electron Optics*. 118(10): 502–506.

[18] Hao Yan, Xiang Peng, Xiaxiang Lin, Wei Jiang, Tian Liu, Hong Guo. 2009. Efficiency of Winnow Protocol in Secret Key Reconciliation. World Congress on Computer Science and Information Engineering, IEEE. 3: 238–242.

[19] Gilles Van Assche. 2006. *Quantum Cryptography and Secret-key Distillation*. New York: Cambridge University Press.

[20] Wuthigorn Traisilanun, Keattisak Sripimanwat, and Ornlarp Sangaroon. 2007. Secret Key Reconciliation Using BCH Code in Quantum Key Distribution. International Symposium on Communications and Information Technologies.

[21] David Elkouss, Anthony Leverrier, Romain All´eaume, and Joseph J. Boutros. 2009. Efficient Reconciliation Protocol for Discrete-variable Quantum Key Distribution. *ISIT 2009*, Seoul Korea, June 28 - July 3, 2009.

[22] Moreira, J. C., and Farrell, P. G. 2006. *Essentials of Error-Control Coding*. England: John Wiley & Sons.

[23] Neubauer A., Freudenberger J., and Kuhn V. *Coding Theory: Algorithms, Architectures and Applications*. 2007. England: John Wiley & Sons.