Improving Throughput in BB84 Quantum Key Distribution

Shawn Prestridge and James Dunham
Electrical Engineering, Southern Methodist University, 6425 Boaz Ln, 75205, Dallas, Texas, U.S.A.

Keywords: QKD, BB84, Winnow, B92, CASCADE.

Abstract: Quantum Key Distribution (QKD) is a scheme that allows two parties to exchange a key in a provably secure manner that will be used in a more conventional encryption system. The first implementation of QKD was BB84 by Bennett and Brassard. Several techniques have been used to attempt to maximize the number of bits realized at the end of the BB84 protocol. One of the techniques used is the B92 protocol (and its follow-on, CASCADE) introduced by Bennett *et al.* which uses 1-bit parities to reconcile the keystream between Alice and Bob (Bennett et al., 1992). Another is the Winnow protocol introduced by Buttler *et al.* which uses Hamming codes to increase the efficiency of the BB84 protocol to allow error rates up to 13.22%(Buttler et al., 2003). In this paper, the Winnow protocol is enhanced and extended to allow arbitrarily high error rates thus greatly improving the effectiveness of the protocol while preserving security. This enhancement also provides a marked improvement over the original B92 protocol in terms of the number of bits preserved in the keystream.

1 INTRODUCTION

BB84 is a well-known protocol to do QKD and quite a bit of research has been done on the last part of the protocol (commonly referred to as the reconciliation phase) which ensures that Alice and Bob have the same bitstream that will be used as a key in their conventional cryptosystem. Most algorithms rely on simple parity-check schemes to ensure that Alice and Bob can detect and correct inconsistencies between their respective bitstreams. Moreover, they also rely on bit-discarding (often referred to as "burning") techniques to restore security to the remaining bits. This paper will briefly describe the manner in which this resolution takes place and will also provide the basic framework for systematic codes that extend the Winnow protocol to be effective at very high error rates.

1.1 Brief Description of BB84 (Bennett and Brassard, 1984)

The BB84 protocol allows Alice to send a new cryptosystem key to Bob in a manner that is provably secure via a Quantum Channel (QC). Alice chooses a key k and a series of orthogonal bases b on which she encodes k so that they can be transmitted across a QC. Both k and b have maximum entropy and the same length for reasons that are detailed in the original paper. Bob chooses a random set of bases b' by which he measures what Alice sends to him. Due to the nature of the orthogonal bases and the QC, if Bob does not guess the correct basis by which to measure a bit of k, then the result becomes random for that bit. At a high level, the protocol can be summarized in a series of steps in $Fig.1$.

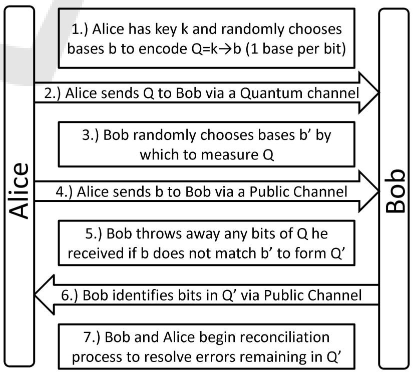
Alice
Bob
1.) Alice has key k and randomly chooses bases b to encode Q=k→b (1 base per bit)
2.) Alice sends Q to Bob via a Quantum channel
3.) Bob randomly chooses bases b′ by which to measure Q
4.) Alice sends b to Bob via a Public Channel
5.) Bob throws away any bits of Q he received if b does not match b′ to form Q′
6.) Bob identifies bits in Q′ via Public Channel
7.) Bob and Alice begin reconciliation process to resolve errors remaining in Q′

Figure 1: BB84 Protocol illustrated.

This paper will focus solely on improving the methods used in the last step of the protocol, so the only security concerns that need to be addressed concern the leakage of key information in the reconciliation process. In the last step, Alice and Bob have approximately the same bitstream which will be dis-

Prestridge, S. and Dunham, J.
Improving Throughput in BB84 Quantum Key Distribution.
DOI: 10.5220/0006422104370443
In *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE 2017) - Volume 4: SECRYPT*, pages 437-443
ISBN: 978-989-758-259-2

tilled down to a final key. There are a number of protocols that are designed to resolve these remaining errors, but the underlying theme to most of these protocols is that that Alice and Bob select a number of bits from their bitstream to form a block and compute a parity check value on the block. They then exchange this parity check value across an unsecured medium (most likely the Internet) to try to determine if there are errors in the block. If there is an error, the protocol attempts to correct the errors. These protocols stipulate that certain bits within the remaining bitstream must be discarded in order to prevent information leakage due to the fact that the parity check information was exchanged on an unsecured medium. Where these methods typically differ from one another is in the way they comprise blocks, compute parities, and when they discard bits. Additionally, some protocols will use Privacy Amplification (PA) which feeds the distilled key into a hash algorithm to increase security. Even if a protocol does not include PA as one of its steps, it can add PA as a bolt-on to its output to increase security by use of randomness extractors (Dodis et al., 2004).

1.2 B92 Protocol (Bennett et al., 1992)

In the reconciliation phase, the B92 protocol divides the bitstream into blocks of length k (the paper provides empirical data on how to determine the length of k) and then computes a 1-bit parity on each block. Each block has a random selection of bits from the bitstream with each bit only going into one block per round to ensure that all bits are checked. When the parity information is exchanged, a bit is burned from the block to preserve security. If the parities agree, then they move on to the next block. If the parities between Alice and Bob disagree, they split the block in half and perform a 1-bit parity on each half of the block, burning a bit on each half to preserve security; this process is repeated until the error in the block is eliminated. When all blocks have been checked, the process is repeated again with the smaller bitstream (because bits have been discarded along the way to preserve security). These rounds of error-detection with 1-bit parities are repeated until 20 successive rounds have been completed with no errors found in any of the blocks. At the end of this reconciliation, the remaining bits can be used as-is or can be fed into a PA scheme.

1.3 Winnow Protocol (Buttler et al., 2003)

The reconciliation phase of the Winnow protocol computes a simple 1-bit parity for the block and if the parities computed by Alice and Bob match, then they assume that the block does not contain an error and a single bit is discarded from each of their blocks to restore maximum entropy to the remaining bits in the block. If the parities do not match, the block is bisected and a 1-bit parity is computed for each half of the block to determine in which half the error lies. This protocol sacrifices a bit for every parity bit that is exchanged on the unsecured channel, therefore it preserves security in a fashion that most closely resembles the one that will be presented in this paper and will thus be the primary baseline for comparison. Once this single-bit parity round is complete, Winnow uses Hamming codes to attempt to further eliminate errors and preserve information-theoretic security. Winnow is typically compared to the CASCADE protocol and B92 for efficiency purposes (Bennett et al., 1992). CASCADE delays the burning of bits but uses privacy amplification to address the issue, therefore it is better to compare our new protocol to Winnow and to B92.

2 PDG CODES

This paper will introduce a protocol called Prestridge-Dunham Guardian (PDG) Codes that will further the Winnow protocol by exploiting probabilities concerning the Hamming code syndrome and the number of estimated errors remaining in code blocks. PDG codes will allow them to resolve errors in the bitstream with a much greater frequency than Winnow and at much higher bit error rate (BER) values. PDG Codes consist of a series of rounds that iteratively search through the string identifying and correcting errors. After several rounds, Bob's string will then be compared with Alice's to see if they are indeed equivalent.

In general, the PDG Coding System will mirror Winnow by taking the entire message that was transmitted from Alice to Bob and break it up into a series of blocks of a particular length (the specifics of which length to choose are detailed in section 2.2). Where PDG will differ from Winnow and how it achieves much higher error detection and correction rates is in how the syndrome information is exploited to maximize the error detection and correction capability. In both PDG and Winnow, the following takes place for each block:

1. Alice computes a Longitudinal Redundancy Check (LRC) of the block (essentially a one-bit parity check) and sends the result to Bob;
2. Bob computes the LRC for his received block;
3. If Bob's computation matches that of Alice, they assume that the bits in that block have no errors and sacrifice one bit in the block to re-randomize the data;
4. Otherwise, Alice and Bob independently compute Hamming Code parity bits and Alice sends hers to Bob via the unsecured public channel; Bob uses this information to correct his received data and he and Alice sacrifice as many message bits as there were parity bits exchanged on the unsecured channel.

Since the LRC is a linear systematic code, sacrificing a message bit in the block will re-randomize the data to account for the single parity bit that was exchanged on the unsecured channel; this result has been proven many times (Prestridge, 2017). As PDG progresses, it will always sacrifice as many message bits as there were parity bits exchanged on the public channel, so at each step along the way the data is re-randomized in such a way that Eve gleans no information about the remaining bits. Hamming codes were selected for that very reason: the number of parity bits that it uses are low thus providing a good throughput of message bits. Other error-detection and -correction codes can be used, but they will require delaying the sacrificing of message bits to preserve message bit throughput. Therefore, there will be some amount of information leakage before a PA scheme is used to distill a final key. As it stands, PDG gives the user the option whether they want to use PA or not since the bits it distills are secure. Once all the blocks have been processed, it is likely that Alice and Bob will engage in a second round of resolution with a larger block size to account for the fact that the number of errors in the message bits has decreased, thereby gaining performance for the second round. The real efficiency of PDG comes from how Hamming codes are exploited to get better results than other codes.

2.1 Getting More from Hamming Codes

To see how we can more effectively exploit the syndrome, let us first examine a very simple Hamming code. A standard [7,4] Hamming code has seven overall bits in the codeword, of which four are information and three are parity bits. The mechanics of how to compute the syndrome for single-error detection and correction can be found in many textbooks (MacWilliams and Sloane, 1978). A standard syndrome for decoding single-bit errors is:

$$\begin{bmatrix} s_3 & s_2 & s_1 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} \text{Bit in error} \\ \hline \text{No Error} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{bmatrix} \tag{1}$$

Using the [7,4] code, four information bits are being transmitted over the noisy secure (quantum) channel and the three parity bits are being transmitted over a noiseless insecure channel (Internet). It is known that the Hamming distance of this code will allow the user to correct all single-bit errors, but what about correcting double-bit errors since it is now known that an error cannot occur in the parity bits? A standard Information Theory maxim is that any error rate at or below $1E^{-6}$ is considered negligible (MacWilliams and Sloane, 1978). Since the parity information is transmitted on the Internet which has a BER less than $1E^{-6}$, the parity information can be considered a lossless transmission as the IEEE 802 Fuctional Requirements Document provides that the error rate on the MAC Service Data Unit is below this threshold (IEEE, 1991). A traditional analysis of the [7,4] Hamming code says that if there is more than a single bit error per packet, the error is undetectable and uncorrectable. For a 5% BER, the probability of an undetected error is given by:

$$\sum_{i=2}^{4} \binom{4}{i} p^i (1-p)^{4-i} = 0.0140. \tag{2}$$

Thus, the probability of an undetected error is approximately 1.40%.

Because the parity bits are transmitted perfectly, if the syndrome indicates that an error has been encountered in one of the parity bits it can be logically assumed that what has actually happened is that more than one bit error has occurred. Because of the symmetric nature of Hamming codes, it is only necessary to examine one case of the pool of possible information bit combinations, so vector [0000] is used. Table 1 shows that six different double-bit errors can occur within the four information bits.

Table 1: Syndromes for all single-bit errors.

Received vector	Bit errors	Syndrome
0010100	3, 5	110
0010010	3, 6	101
0010001	3, 7	100
0000110	5, 6	011
0000101	5, 7	010
0000011	6, 7	001

In the case of a double-bit error, there are two possibilities: the resultant syndrome indicates a single bit error in either a parity bit or a single-bit error in a non-parity bit. When the syndrome indicates that an error has occurred in a non-parity bit (double errors in code bits [3, 5], [3, 6] and [5, 6]), an undetectable error has occurred. However, three of the six possible double-bit error patterns indicate an error in a parity bit ([3, 7], [5, 7] and [6, 7]); since we know that a parity bit error cannot occur, then we know that one of these three double-bit error patterns has occurred. Three of the four information bits will be discarded, so the user only needs to be concerned about correcting the bit that will be kept. Therefore, there are really only three possible double-bit error patterns that concern the user and so no matter which bit the user selects to keep, there is one double-bit error pattern that is detected and corrected, thus improving the [7,4] Hamming code by lowering the probability of an undetected error to 0.95%:

$$\frac{2}{3}\binom{4}{2}p^2(1-p)^2+\sum_{i=3}^{4}\binom{4}{i}p^i(1-p)^{4-i} = 0.0095066. \tag{3}$$

The chosen bit will still be correct half of the time, so the effective error rate is approximately 0.475%, a 29.52% improvement over the traditional method.

This same technique can be extrapolated to larger Hamming codes, *e.g.* consider a block that is being checked with a (31,26) Hamming code and Bob's computed syndrome is 00100. There are eleven different double-bit error patterns that map to the same syndrome. Since there were five parity bits p_j that were exchanged across the unsecured public channel, five bits of the block b_j must be burned to preserve the security. If Bob chooses one bit from five of the eleven possibilities, he has a $\frac{5}{11} = 55.6\%$ chance of eliminating one of the bits that was in error. Moreover, if Bob was successful in eliminating one of the errors in the block, then it is easier to eliminate the other error in the block on a different pass.

2.2 Detailed Outline of the PDG Coding System

PDG begins by looking at both k and k' to remove the remaining errors. In most QKD schemes, both Alice and Bob have an estimate of the BER for their channel because they run a preliminary string of data through the QC to estimate the BER. With this BER in mind, the PDG code proceeds like this for each round:

1. Choose block length k to divide the string into k-length blocks, zero-padding the last block to make it fit the proper length.
2. Randomly select bits from the string to fill the block; each bit will be selected only once per round.
3. Compute a one-bit LRC check for the block; if the LRC check passes, burn one bit and move to the next block.
4. If the LRC fails, use a Hamming code in the manner described in section 2.1 to eliminate the error; burn $(n-k)$ bits as this is how many parity bits were used in the Hamming code.
5. Repeat until all blocks have been processed for this round.

The number of rounds necessary have been determined experimentally for different initial BERs. Now we will delve into the details for each step mentioned above.

2.2.1 Choosing Block Lengths

Since the BER (p_{BER}) on the QC is known, we can compute the probability of having more than one bit error in the block via the formula:

$$p_{UDE} = \sum_{i=2}^{k}\binom{k}{i}p_{BER}^{i}(1-p_{BER})^{k-i} \tag{4}$$

where p_{UDE} is the probability of an undetected error for a Hamming code. We adjust our block length such that the p_{UDE} is approximately 1% to get maximum efficiency for our code by balancing the need to burn bits but still having a high-rate Hamming code. There are only certain Hamming codes available (*e.g* (7,4), (15,11), *etc.*), so the selection of k must take this into account and err on the side of caution so that results will be skewed conservatively.

2.2.2 Randomly Selecting Bits to Fill Each Block

Assuming that an (n,k) Hamming code is chosen, the string will be broken up into a series of k-length blocks, the last block being 0-padded if necessary. These blocks are filled with bits randomly selected from the distilled bits of Alice and Bob's respective collections; they each select the same random pattern from their collections to fill blocks and each bit goes in precisely one block for the round. The rationale for this mixing is to prevent "bursty" errors (a string of successive bit errors) from corrupting the entire string which would burn more bits than is necessary.

2.2.3 Computing the LRC on the Block

Once the blocks are filled, Alice and Bob will run a Longitudinal Redundancy Check (LRC) on each block b_j. The results are compared across the unsecured public channel which allows Alice and Bob to do a rudimentary check for an odd number of bit errors without burning a great deal of information bits. If the LRC does not detect an odd number of errors, then the block is considered to be free of errors because the careful selection of the block size minimizes the probability that there are two or more errors contained within a single block. Clearly, this can sometimes be incorrect but these errors are almost always caught by running successive rounds of the algorithm.

2.2.4 If LRC Fails, Run Hamming Code

If the LRC indicates the presence of an odd number of errors, Bob will ask Alice over the unsecured public channel for the parity bits p_j of the Hamming code for the block being examined. The block b_j is combined with those parity bits to form c_j and then multiplied by the corresponding parity check matrix H to generate the syndrome s:

$$c_j \bullet H = s. \tag{5}$$

This syndrome is then used in the manner previously described to determine if Bob's b_j has no errors, a single error or two errors. If two errors are detected, Bob uses the syndrome tables to determine which combination of bits in b_j can produce the computed syndrome. Depending on which (n,k) Hamming code is currently being used, it may be possible for Bob to correct one or both errors; if not, Bob earmarks all the suspect bits as being "sacrificable" in order to minimize the number of times the suspect bits are checked. This process is continued for each block b_i in the string.

For each block, Bob must burn as many bits in b_j as the number of parity bits p_j that were communicated on the unsecured channel. If a single-bit error has been detected, then Alice sends Bob the parity information for the corresponding Hamming Code. If a double-bit error has been detected, Bob will examine the number of possible pairs of bits in b_j that could generate such an error and select one bit from separate pairs to burn; this increases the probability that Bob has removed at least one of the errors in the block without burning a bit in the rest of b_j that has a high probability of being correct.

2.2.5 Repeat for all Blocks

This process is repeated for all the blocks remaining in the round. When all the blocks have been checked, Bob uses the unsecured public channel to send a list of the bits that he has burned so that Alice will also remove those bits from her pool. Again, knowing the position of the bits that have been burned does not compromise the security because the parity of the remaining bits in the sundry b_j blocks are rerandomized which prevents Eve from gleaning any information about them.

The number of rounds and which Hamming codes to use have been determined experimentally and are summarized in Table 2 for BERs of 1%-10%.

Table 2: PDG Selection parameters by BER.

BER	Pass 1	Pass 2	Pass 3	Pass 4
1%	(15,11)	(31,26)	(63,57)	(127,120)
2%	(15,11)	(31,26)	(63,57)	(127,120)
3%	(15,11)	(31,26)	(63,57)	(127,120)
4%	(15,11)	(31,26)	(63,57)	(127,120)
5%	(15,11)	(15,11)	(63,57)	(127,120)
6%	(15,11)	(31,26)	(31,26)	(31,26)
7%	(15,11)	(15,11)	(31,26)	(31,26)
8%	(15,11)	(15,11)	(15,11)	(31,26)
9%	(15,11)	(15,11)	(15,11)	(31,26)
10%	(15,11)	(15,11)	(15,11)	(15,11)

The bits that remain can be used either as a key for a cryptosystem or they can be used in a PA scheme. For comparison, we look at B92 and Winnow since they give the number of raw bits remaining before a PA stage is applied.

3 RESULTS

We will now compare PDG against two of the most commonly used techniques for performing bit reconciliation, namely B92 and Winnow.

3.1 PDG *vs.* B92

On average, the PDG codes return better results 80% of the time than the B92 protocol because either:

1. PDG returns more bits to Alice and Bob than B92.
2. PDG corrects all errors in the block and B92 does not.

Clearly, the PDG codes demonstrate a remarkable improvement over B92 while still preserving the security of the key exchanged by the protocol.

Running a slate of tests that allow the BER to vary from 1% to 10% and the initial message size to vary from 1k to 10k bits (a total of 100 test parameters with

10k tests per parameter) reveals some surprising results for the B92 protocol. While PDG maintains approximately the same performance for a given BER regardless of the initial packet size, the performance of the B92 protocol suffers drastically as the size of the packets increases for all BERs greater that 2%. Selecting typical use cases of a BER of 1%, 3% and 7%, it is easily seen that the PDG code outperforms the B92 protocol on almost all of the test cases performed. Given that the PDG performance is equivalent regardless of the packet size (and to allow B92 to perform its best), a packet size of 1,000 bits can be compared. For the 1% case, PDG returns a total of 252,095 bits more than B92, a 3.1% improvement. For the 3% case, PDG returns a total of 200,443 bits more than B92 for a 2.8% improvement. For the 7% case, PDG returns a total of 2,466,405 bits, an 83.9% improvement. From these cases, it is clear that PDG provides a more efficient coding solution than B92 for extracting bits in a QKD scheme. For the grand total of 1,000,000 tests run to compare how PDG performs against B92, PDG was able to outperform B92 more than 50% of the time on 99 of the 100 parameters and missed a perfect record by a mere 4% on the test parameter in question.

Table 3 shows a typical use case where Alice and Bob would exchange 2,000 bits. Each row of the table shows the results of running 10k tests with the selected BER and 2k packet size. The column for "% Int" is the percentage of results that were deemed interesting, *i.e.* where PDG successfully returns bits and either B92 does not return as many bits or B92 does not remove all the bit errors. The columns "PDG Bad" and "B92 Bad" indicate the number of test cases where PDG and B92 did not remove all the errors, respectively. The column of "Bit diff" shows the total number of bits that PDG leaves in excess of those left by B92 when both PDG and B92 were able to eliminate all errors in the packet.

Table 3: Comparing PDG to B92.

BER	% Int	PDG Bad	B92 bad	Bit diff
10	84.01	1,370	8,856	6,634,518
9	85.15	1,240	8,864	7,446,470
8	90.12	304	8,868	8,777,072
7	89.04	310	8,750	9,509,198
6	68.19	1,679	8,125	7,700,175
5	58.40	2,130	6,044	5,315,023
4	70.81	2,417	3,050	1,110,910
3	93.06	421	822	873,984
2	99.17	37	110	419,612
1	100.0	0	10	304,615

PDG returns a successful result at least 75% of the time and quite often more than 85% of the time. When you divide the total number of PDG bits by the number of test cases run (10000), you can see that the average number of bits for each test would leave an adequate key length regardless of which standard cryptosystem was employed by Alice and Bob for their communication (at 10% BER, an average of 759 bits per case and at 1%, an average of 1685 bits per case). It is interesting to note that PDG leaves more bits than the B92 in all cases, particularly when the BER is high. At a high BER, B92 can rarely correct all errors in Bob's bitstream. As the physical distance between Alice and Bob increases, so does the BER on the quantum channel. These results indicate that PDG would permit a greater physical distance to exist between Alice and Bob than could be tolerated by B92.

3.2 PDG *vs.* Winnow

Winnow is capable of distilling a key so long as the BER on the quantum channel is below 13.22% (Buttler et al., 2003). However, PDG is capable of returning keys for error rates up to 47%. PDG effectively has some double-error detecting and correcting capability that Winnow does not; as such, PDG is capable of distilling at least some key between Alice and Bob when Winnow cannot. As one would expect, the probability of successfully distilling a key and the length of the distilled key become arbitrarily low as the BER increases and the size of the initial packet decreases. Table 4 shows the effectiveness of PDG at a 47% error rate when the initial packet size varies from 10000 bits down to 1000 bits. In this table, the "% Int" column shows the number of times PDG was able to correct all errors in the packet and "PDG Bad" is the number of cases where PDG could not successfully remove all errors in the string; "Ave. bits" is the average number of bits left when PDG successfully removed all errors.

Table 4: PDG performance at 47% BER.

BER	Size	% Int	PDG Bad	Ave. bits
47	10000	99.39	61	204
47	9000	99.25	75	183
47	8000	99.01	99	162
47	7000	98.34	166	141
47	6000	97.79	221	120
47	5000	97.12	288	99
47	4000	95.18	482	79
47	3000	92.91	709	58
47	2000	89.43	1,057	38
47	1000	80.60	1,940	19

Since PDG is capable of consistently correcting all errors even at high error rates, it can be used to quickly distill a secret key between Alice and Bob. As the sample table indicates, the performance at a 47% BER is roughly linear with a 1k bit packet returning an average of 19 bits per case (1.9% of the bits make it through the distillation) and a 10k bit packet returns an average of 204 bits per case (2.0% of the bits make it through the distillation).

Comparing PDG to Winnow at rates below 10% further illustrates the efficiency gain realized by PDG. Table 5 shows the average fraction of bits remaining at bit error rates ranging from 1% to 10% when Winnow is used with the PDG parameters specified in Table 2. These results reflect only the error-detection and -correction portion of the respective codes without PA; assuming that the same PA would be applied to both codes, the final results including PA would be multiplied by the same scalar.

Table 5: Fraction of bits remaining at various error rates.

BER	Winnow remaining	PDG remaining
10	0.1640	0.3866
9	0.2081	0.4307
8	0.2152	0.4943
7	0.2642	0.5450
6	0.3353	0.5097
5	0.3412	0.5463
4	0.4331	0.5864
3	0.4563	0.7361
2	0.5211	0.8041
1	0.6121	0.8426

4 CONCLUSIONS

PDG codes are capable of outperforming the most popular BB84 coding schemes by utilizing extra information available from Hamming codes. This enables PDG to leave more bits than the B92 protocol more than 80% of the time while not leaking any information bits to Eve. Additionally, PDG can outperform the Winnow protocol by being able to reliably detect and correct errors all the way up to 47% BER on the quantum channel, thus allowing Alice and Bob to create longer fiber links between them to take advantage of the enhanced throughput. Even when the BER is less than Winnow's maximum, PDG leaves more bits to create a longer key than Winnow can distill.

REFERENCES

Bennett, C. H., Bessette, F., Brassard, G., Salvail, L., and Smolin, J. (1992). Experimental quantum cryptography. *Journal of Cryptology*, 5(1):3–28.

Bennett, C. H. and Brassard, G. (1984). Quantum Cryptography: Public Key Distribution and Coin Tossing. In *Proceedings of the IEEE International Conference on Computers, Systems and Signal Processing*, pages 175–179, New York. IEEE Press.

Buttler, W. T., Lamoreaux, S. K., Torgerson, J. R., Nickel, G. H., Donahue, C. H., and Peterson, C. G. (2003). Fast, efficient error reconciliation for quantum cryptography. *Phys. Rev. A*, 67:052303.

Dodis, Y., Reyzin, L., and Smith, A. (2004). *Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data*, pages 523–540. Springer Berlin Heidelberg, Berlin, Heidelberg.

IEEE (1991). Functional requirements IEEE 802. http://www.ieee802.org/802˙archive/fureq6-8.html. Accessed: 2017-05-21.

MacWilliams, F. and Sloane, N. (1978). *The Theory of Error-Correcting Codes*. North-Holland Publishing Company, 2nd edition.

Prestridge, S. (2017). *Applying Classical Information Theory to Quantum Key Distribution*. PhD thesis, Southern Methodist University. unpublished thesis.