# *LDPC error correction in the context of Quantum Key Distribution*[*]

Anastase Nakassis and Alan Mink
Information Technology Laboratory
National Institute of Standards and Technology,
100 Bureau Dr., Gaithersburg, MD 20899
anakassis@nist.gov, amink@nist.gov

## Abstract

Secret keys can be established through the use of a Quantum channel monitored through classical channel which can be thought of as being error free. The quantum channel is subject to massive erasures and discards of erroneously measured bit values and as a result the error correction mechanism to be used must be accordingly modified. This paper addresses the impact of error correction (known as Reconciliation) on the secrecy of the retained bits and issues concerning the efficient software implementation of the Low Density Parity Check algorithm in the Quantum Key Distribution environment. The performance of three algorithmic variants are measured through implementations and the collected sample data suggest that the implementation details are particularly important

**Keywords:** Quantum key distribution, QKD, Entropy, Renyi Entropy, LDPC

## 1. INTRODUCTION

The Quantum Key Distribution (QKD) protocol [1,4,9,13,14,17,18,22,24,25] uses a quantum channel and a classical channel to establish an information theoretically secure secret that is shared by two parties, Alice and Bob. Information theoretically secure means that the protocol will produce a **k** bit string, **s**, so that the Renyi entropy of **s**, **R(s)**, as seen by an outside observer, exceeds $\mathbf{k}-\boldsymbol{\varepsilon}$ with likelihood exceeding $\mathbf{1}-\boldsymbol{\delta}$, $\boldsymbol{\varepsilon}$ and $\boldsymbol{\delta}$ being arbitrarily small and positive preselected algorithm constants. **R(s)** is a measure of secrecy so that a reduction of secrecy is implied through a decrease in **R(s).**

The narrative on QKD has evolved over the years as more sophisticated attacks [14,18] and unintended information leakages in the interactive Reconciliation algorithms [17] have been discovered, but the essentials remain the same and a high level functional description of the QKD protocol follows:

1) Through a public dialogue Alice and Bob will retain a sequence of **N** bit values, known as sifted bits which, in the absence of noise and eavesdropping, would have been identical and totally secret; the estimated error frequency, known as the quantum bit error rate (QBER), is a measure of the potential interference and eavesdropping in the quantum channel and can provide, with likelihood close to 1, a lower bound on the secrecy of the initial part of the QKD protocol.
2) Alice and Bob engage in an additional public dialog that attempts to correct whatever errors are present between their sifted bits; this process will expose some additional information about the sifted bits and will further reduce their secrecy.
3) Finally, and with likelihood close to 1, Alice and Bob are left with two identical strings of **N** bits. Provided that sufficient secrecy remains, there are hash functions that can map **s** onto a shorter string, **s***, of length **k**, in such a way that with likelihood close to 1, the secrecy of the resultant string is arbitrarily close to **k**.

[*] The identification of any commercial product or trade name does not imply endorsement or recommendation by the National Institute of Standards and Technology.

The drop in secrecy in steps 1) and 2) above is O(**N**) and roughly speaking no secret will be left if the QBER exceeds 11%. The cost, in bits, of ensuring that we shall not overestimate the string's secrecy in steps 1) through 3) is O(-log($\boldsymbol{\delta}$)) and the penalty, in bits, for producing a bit-string whose secrecy is (with likelihood close to 1) within $\boldsymbol{\varepsilon}$ of its length is O(-log($\boldsymbol{\varepsilon}$)). In short, if there is a positive constant $\boldsymbol{\mu}$ such that (with likelihood close to 1 and for big **N**) the Renyi entropy of **s** after step 2) exceeds $\boldsymbol{\mu}$**N**, with likelihood close to 1, meaningful secrets can be produced.

We aim at providing a stricter, than the one usually cited [7,26], estimate of the drop in secrecy during the Reconciliation process and at providing a blueprint for efficient implementations of the Low Density Parity Check (LDPC) algorithm within the context of QKD Reconciliation.

The QKD Reconciliation process has gradually evolved from interactive algorithms [2,3], such as Cascade, to LDPC based algorithms [11,12,15,19,20]. This evolution to LDPC has been primarily for reasons of performance, but it should also be noted that interactive error correction can result in unintended covert channels, releasing information that is hard to quantify and often not properly accounted for [18]. Nevertheless, relatively little is known about the details of the implementations of LDPC algorithm in the QKD context and the straightforward adaptation of the published algorithms reveals unexpected difficulties.

This paper will briefly discuss the impact of Reconciliation on the Renyi entropy of the string, **s**, and then it will cover efficient algorithmic variants of the LDPC error correction schemes that are attuned to QKD and can be easily mapped into executable code. We present three algorithmic variants of which the first is based on real arithmetic while the other two use integer arithmetic that is amenable to fast table lookup schemes. Finally, we present a snapshot of the performance of these implementation versions.

## 2. THE IMPACT OF RECONCILIATION ON EVE'S INFORMATION

In what follows we assume that the N-bit vectors **x**={$x_1$, $x_2$, … , $x_N$} and **y**={$y_1$, $y_2$, … , $y_N$} represent, respectively, the sifted bits of Alice and the sifted bits of Bob; Eve's knowledge of **x** is encapsulated in a random variable **Z**={ $Z_1$, $Z_2$, … $Z_N$}. Vector **y** is related to **x** by **y = x** XOR **d**, where **d**={$d_1$, $d_2$, … ,$d_N$} is Bob's unknown, **N**-bit error vector whose non zero entries correspond to the errors in Bob's measurements. The weight of **d**, **|d|**, the number of non-zero entries, is an indication of the severity of Eve's attack and can be used to establish likelihood bounds on Eve's knowledge, i.e., R(**Z**). An estimate of **|d|**, **d***, will allow us to bootstrap the Reconciliation process. Such an estimate can be obtained through sampling (of bit values other that the ones being reconciled) or by monitoring the error frequency in previous reconciliation batches. The estimated QBER, **d*/N**, will determine the choice of the LDPC Gallager matrix.

Several papers [e.g., 5,6,7,21] address the impact of error correction on **R(Z).** Some address asymptotic properties whose impact on real systems is not entirely clear. Moreover, much of the discussion is based on the assumption that the quantum channel is thin (slow) and the objective of Reconciliation is to release as little information as possible. Nevertheless, there are test-beds that can sustain quantum channel rates exceeding a GHz yielding sifted data at mega-bits per second [16]. Therefore, the appropriate criterion for a Reconciliation algorithm is to maximize a system's throughput rate (privacy amplified output) and not necessarily operating near the Shannon limit. E.g., in [15], page 561, it is suggested that up to 200, or even 1000, iterations of the LDPC algorithm may be an acceptable price to pay for error correction, a situation that we would rather avoid.

Let **U** be a discrete random variable over some set **E** and let **R** be the Renyi entropy of **U**. Let **f** be a function over **E** that allows at most $\mathbf{2^k}$ values (i.e., **f**(**U**) can be thought to be a **k**-bit nonnegative integer). Then with probability exceeding $1-2^{-u}$, knowledge of **f**(**U**) will lower **R** by no more than k+2u, where u is some arbitrarily selected value to achieve the desired probability bound.

Let $\mathbf{E_i}$ consists of all the values of **U** for which **f(U)**=i, i=0,1, 2, … , $2^k$-1, let $\mathbf{q_i}$ be the probability that **U** is in $\mathbf{E_i}$, let **T** be the set of i values for which $\mathbf{q_i}$>0, and let $\mathbf{R_i}$ be the Renyi entropy of **U**, when **U** is restricted over $\mathbf{E_i}$ (provided $\mathbf{q_i}$ >0), then

$$2^{-R} = \sum_{i \in T} q_i^2 2^{-R_i}$$

If **V** is the set of all indices in T for which $\mathbf{R_i \leq R\text{-}k\text{-}2u}$, then,

$$2^{-R} \geq \sum_{i \in V} q_i^2 2^{-R+k+2u} \quad \text{and} \qquad 2^{-k-2u} \geq \sum_{i \in V} q_i^2$$

Elementary calculus considerations show that when the sum of the squares of |V| real variables is constant, their sum is maximized when they are equal to each other and positive. In our case this translates into

$$q_i = \sqrt{2^{-k-2u} / |V|}$$

for all i in V, hence

$$\sum_{i \in V} q_i \leq |V| \sqrt{2^{-k-2u} / |V|} = \sqrt{2^{-k-2u} |V|} \leq \sqrt{2^{-2u}} = 2^{-u}$$

In other words, the probability that the LDPC **k**-bit information exchanged will lower the entropy of **x** by more than **k+2u** cannot exceed $2^{-\mathbf{u}}$.

Let us assume that the random variable actually is $\mathbf{Z^*}$=($\mathbf{Z}$, $\mathbf{d^\#}$) where **Z** is Eve's **N** bit vector view of Alice's vector and $\mathbf{d^\#}$ is Eve's view of the number of bits over which Alice's and Bob's bit values differ. Clearly R($\mathbf{Z^*}$)≥R(**Z**), Moreover, in the process of Reconciliation Alice will compute and make public the **M**-bit checksum that is formed from $\mathbf{Gx^T}$ (**G** being the Gallager matrix in use) and the **H** bits for the hash(**x**); Bob will release the weight of **d**, |**d**|, (unless the LDPC correction fails or the hash values of **x** and the corrected **y** do not coincide) as a **u**-bit integer ($\mathbf{u} \leq \log_2(\mathbf{N})+1$). Once this information becomes public the probability distributions of **Z** and **Z*** will collapse on the same values and the residual entropy of **Z** or **Z*** will exceed R(**Z**)-**M**-**H**-**len**-2**u** bits with probability exceeding $1\text{-}2^{-u}$ (i.e., the entropy is reduced by the number of reconciliation bits made public, **M+H+u**, and a safety factor, **u**).

## 3. THE LDPC ALGORITHM FOR QKD

An excellent tutorial with a very readable explanation of the whys of the LDPC related belief propagation algorithm is [23]. Moreover, a multitude of papers, such as [8,10,11,15] describe outlines of the LDPC algorithm in ways that can be directly mapped into an implementation. Nevertheless, unlike the case in communications, the error-correction information is neither synchronous with the bit-values to be corrected nor does it travel over an error-prone channel. The classical channel of a QKD system provides message integrity protection and origin authentication services; as a result it can be assumed to be, for all practical purposes error-free. Thus the QKD version of LDPC will be simpler and for every bit-error-rate will require fewer correction bits than its communications counterpart.

A QKD reconciliation algorithm based on LDPC, that uses the reliable classical channel, can be outlined as follows:

1) Bob and Alice have an estimate of the QBER for the bits to be reconciled
2) Bob and Alice choose the same appropriate **M×N** Gallager matrix, **G**, based on the estimated QBER. Alice sends Bob an **M**-bit check sum vector, C=$\mathbf{G}X^T$ ($X^T$ is the transpose of X and the arithmetic is mod 2), and an **H**-bit hash value, **hash(X)**.
3) Bob computes a check sum vector, D=$\mathbf{G}Y^T$. If C≠D, we iteratively apply the LDPC belief propagation algorithm to obtain Y', a new estimate of the corrected Y bits, recomputing D'=$\mathbf{G}Y'^T$ on each iteration, until C=D' or the maximum iterations have been reached.
4) If C≠D', the error correction has failed; otherwise hash(Y') is compared to **hash(X)** to verify, with great likelihood, that X=Y'. If C≠D' or if the hash values do not agree, Bob sends Alice a failure message which is to be followed by whatever failure-handling facilities are at hand; e.g., Alice and Bob can release their bit values for a post-mortem.
5) If no failure has been observed, Bob sends Alice a message indicating success which may also include a count of the bits in error, i.e. |**d**|**;** this count can be used to update the system's estimate of the QBER.

Each iteration of the LDPC belief propagation algorithm has two steps: The first, in its simplest form, operates on and returns a-values; each of them is of the form **a=1-2p**, p being an estimate that the bit was received in error. The second step, in its simplest form, operates on and returns likelihood f-values of the form **f=(1-p)/p**. The a and f values for the same p are linked through the equations **f=(1+a)/(1-a)** and **a=(f-1)/(f+1)**.

The LDPC algorithm requires access to the following resources:

1) A Gallager matrix, **G**, of dimension **M×N**; **L** is the number of nonzero entries in **G**;
2) **L** registers, one for each nonzero **G** matrix entry. If the (k,i) entry of G equals 1, and if this entry is the ja-th non-zero entry of its row and the jb-th non-zero entry of its column, then $\mathbf{R}_{k,,ja}$ and $\mathbf{r}_{i,jb}$ resolve to the register corresponding to the (k,i) entry of G.
3) A sign vector **v** of dimension **M** such that **v[k]**=1 if **C[k]**=**D[k]**, otherwise **v[k]**= -1 for k=1,2,…,M;
4) A vector **t** of dimension **M;** t(k) is the number of nonzero entries of row k in G.
5) A vector **m** of dimension **N**; m(i) is the number of nonzero entries of column i in **G**.

In what follows we use an asterisk in the superscript position of the input registers when we define a group of transforms in which the input and the output registers overlap; the asterisk indicates that the input values must be retrieved out of temporary storage where copies of the input registers' values have been stored.

With this in mind, the LDPC algorithm is initialized as follows:

- All belief registers, $\mathbf{R}_{k,i}$, are initialized with the a-value 1-2*QBER.
- The elements of **v** are initialized (**v[k]**=1 if **C[k]**=**D[k]**, otherwise **v[k]**= -1, **k**=1,2, … , **M);**
- The symbol λ is the **f**-value of the batch's likelihood ratio, λ= (1-QBER)/QBER.

Each algorithm iteration proceeds then as follows:

**Step1**: For each checksum k, k=1,2,…,M, collects the registers ({$\mathbf{R}_{kj}$| j=1,2,…,t}), t=t(k), and updates their values as follows:

$$R_{k,j} = v_k \prod_{1\le i\le k}^{i\ne j} R^*_{k,i} \quad j=1,2,\dots,t(k)$$

**Step2**: For each bit i, i=1,2,…,N

a) map each of its m=m(i) belief registers, $\mathbf{r}_{i,j}$, from its current a-value representation to the corresponding f-value representation by:

$\mathbf{r}_{i,j}$**=(1+** $\mathbf{r}_{i,j}$**)/(1-** $\mathbf{r}_{i,j}$**)** j=1, 2, … , m(i)

b) then compute

$$F_i = \lambda \prod_{1\le j\le m(i)} r_{i,j}$$

and form vector Y' , where $Y'_i = 1-Y_i$ if $F_i<1$ otherwise $Y'_i = Y_i$

c) then update the likelihood f-values in the $\mathbf{r}_{i,j}$-registers as follows:

$$r_{i,j} = \lambda \prod_{1\le k\le m}^{k\ne j} r^*_{i,k} \quad j=1,2,\dots,m(i)$$

and then map the f-values to their corresponding a-values through the transform below:

$\mathbf{r}_{i,j}$**=(** $\mathbf{r}_{i,j}$**-1)/(** $\mathbf{r}_{i,j}$**+1)** j=1, 2, … , m(i)

**Step3**: Compute D'=**G**$Y'^T$. If D'=C the LDPC portion of Reconciliation has converged and completes successfully; else if the maximum number of iterations have been exhausted, complete unsuccessfully; otherwise repeat Steps 1 and 2.

The three most obvious QKD LDPC implementations are the following:

1) An efficient version of the algorithm we just described. I.e., an algorithm that relies on toggling between the a and f values through the transforms f=(1+a)/(1-a) and a=(f-1)/(f+1).

2) A version that relies on efficient log(f) $\leftarrow\rightarrow$ -sign(**a**)*ln(|**a**|) transforms (e.g., use of tables) and stores these log values in the registers; this would transform all multiplications into additions.
3) A version that relies on a single information representation, such as log(f), at the cost of using more complicated computations.

In what follows we will consider and compare an implementation of 1) and two implementations (variants of each other) of 3). The latter two are closely related to the general approach described in [10]. We do not address 2) any further in this paper.

## 4. EFFICIENT REAL NUMBER IMPLEMENTATION

Algorithm A has the following modifications to the algorithm in Section 3:

Step1) For each checksum, k, we compute the product $\mathbf{A}= v_k\mathbf{R}_{k,1}\,\mathbf{R}_{k,2} \dots \mathbf{R}_{k,t}$ , t=t(k). Then we update $\mathbf{R}_{k,j}$ values by

$$\mathbf{R}_{k,j} = (\mathbf{R}_{k,j} + \mathbf{A})/ (\mathbf{R}_{k,j} - \mathbf{A}) ,\ j=1, 2, \dots , t(k)$$

Step 2c) Once $\mathbf{F}_i$, i, i=1, 2, … , N, is computed, proceed to compute the corresponding a-values and store them in the $\mathbf{r}_{i,j}$ registers by

$$\mathbf{r}_{i,j} = (\mathbf{F}_i - \mathbf{r}_{i,j})/ (\mathbf{F}_i + \mathbf{r}_{i,j}) ,\ j=1, 2, \dots , m(i)$$

These minor adjustments result in fewer operations; moreover they are structured in such a way that the requirements for temporary storage (due to the fact that a register value should not change while it is needed for further computations) are minimized.

## 5. AN IMPLEMENTATION ELIMINATING REAL NUMBER ARITHMETIC

The objective of this version is to avoid having to flip back and forth between the "**a**" and "**f**" values. Let $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3, \dots, \mathbf{f}_n$ be a sequence of f-values and let $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \dots, \mathbf{a}_n$, be the corresponding **a**-values and let **v** equal either 1 or -1, as before. We wish to compute $\mathbf{F}_r = (1+\mathbf{A}_r)/(1-\mathbf{A}_r)$, with $\mathbf{A}_r= \mathbf{v}\mathbf{a}_1\mathbf{a}_2\mathbf{a}_3 \dots \mathbf{a}_r$, for **r**=1,2,….,n. We observe that:

1. $\mathbf{F}_1= \mathbf{f}_1$ if **v=1**, otherwise $\mathbf{F}_1= 1/ \mathbf{f}_1$.
2. For **r=2, 3, … , n,** $\mathbf{A}_r= \mathbf{A}_{r-1}\,\mathbf{a}_r$, where $\mathbf{a}_r= (\mathbf{f}_r -1)/( \mathbf{f}_r+1)$,

   $\mathbf{A}_{r-1}= (\mathbf{F}_{r-1} -1)/( \mathbf{F}_{r-1} +1)$, and $\mathbf{F}_r= (1+\mathbf{A}_r )/( 1-\mathbf{A}_r)$.
3. After substituting and simplifying we obtain:

$$\mathbf{F}_r = (1+\mathbf{F}_{r-1}\,\mathbf{f}_r )/( \mathbf{F}_{r-1}+ \mathbf{f}_r) \tag{1}$$

Let $\boldsymbol{\alpha}_r = \ln(\mathbf{F}_r)$ and $\boldsymbol{\beta}_r = \ln(\mathbf{f}_r)$, then we can rewrite Eq (1) as

$$\boldsymbol{\alpha}_r = \mathbf{g}(\boldsymbol{\alpha}_{r-1}, \boldsymbol{\beta}_r )= \ln(1+\exp(\boldsymbol{\alpha}_{r-1}+ \boldsymbol{\beta}_r )) - \ln(\exp(\boldsymbol{\alpha}_{r-1})+ \exp(\boldsymbol{\beta}_r )) \tag{2}$$

The definition of **g** can be recursively extended to n, n>2, arguments. If $\mathbf{B}= \{\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \boldsymbol{\beta}_3, \dots , \boldsymbol{\beta}_n\}$, then

$$\mathbf{g}(\mathbf{B}) = \mathbf{g}(\mathbf{g}(\mathbf{B}-\{\boldsymbol{\beta}_n\}), \boldsymbol{\beta}_n). \tag{3}$$

Note that for all $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$,

$$\mathbf{g}(\alpha,\beta)= \mathbf{g}(\beta,\alpha) \text{ and } \mathbf{g}(\alpha,-\beta)= \mathbf{g}(-\alpha,\beta)= -\mathbf{g}(\alpha,\beta)$$

and that for any u and w

$$\ln(1+e^{u})= u+\ln(1+e^{-u}) \text{ and for } u\geq w,\ \ln(e^{u} +e^{w})= u+\ln(1+e^{w-u})$$

Thus the recursive computation of Eq (1-3) can be easily carried out if we can compute $\ln(1+e^{-x})$, x>=0.

It should be noted that:

- As the semantics of **g** make it clear, the value of **g**(**B**) does not depend on how the elements of **B** are indexed.
- The sequence $|\mathbf{A_r}|$, **r**=1, 2, … , n is strictly decreasing provided that $0<|\mathbf{a_r}|<1$ for all r, hence
- The sequence max{ $\mathbf{F_r}$, $\mathbf{1/F_r}$ }, **r**=1, 2, … , n, is decreasing, and
- The sequence $\boldsymbol{\alpha_r} = \mathbf{ln(F_r)}$, **r**=1, 2, … , n, is decreasing in absolute value. In particular if $\mathbf{B}=\{\boldsymbol{\beta_1}, \boldsymbol{\beta_2}, \boldsymbol{\beta_3}, \dots, \boldsymbol{\beta_n}\}$, then

  $$|\mathbf{g(B)}| \leq \min(|\beta_1|,|\beta_2|, \dots, |\beta_n|) \qquad (4)$$

  and equality holds if and only if at least one of the **β**-values equals 0. (The values in **B** are real numbers; hence the corresponding **a**-values cannot be equal to -1 or to 1, unless some $\mathbf{a_r}=0$ (in which case $\mathbf{f_r}=1$ and $\boldsymbol{\beta_r} = \ln(\mathbf{f_r}) = 0$) and the inequality (4) will be strict)

If we solve Eq. (1) and (2) for $\mathbf{F_{r-1}}$ and $\boldsymbol{\alpha_{r-1}}$ respectively, we obtain:

$$\mathbf{F_{r-1} = (1\text{-}F_r\, f_r\,)/(\,F_r\text{-}\, f_r)}$$
$$\boldsymbol{\alpha_{r-1}} = \mathbf{g^*}(\boldsymbol{\alpha_r}, \boldsymbol{\beta_r}) = \ln|1-\exp(\alpha_r+\beta_r)| - \ln|\exp(\alpha_r) - \exp(\beta_r)| \qquad (5)$$

with the last equation requiring the ability to compute $\mathbf{ln|1 - e^x|}$.

We now have all the tools necessary for an alternative implementation of the belief propagation algorithm of Section 3. This implementation uses a single representation, the ln of f-values, rather than switching between "a" and "f" values as in algorithm **A**. To avoid the complexity of exponentiation and logarithms in the computation of Eq (2 & 5), we use integer lookup tables, thus avoiding real number arithmetic. Furthermore, using logarithms reduces our computations to addition/subtraction rather than multiplication/division. Our L belief registers, referenced by either $\mathbf{r}_{i,j}$ or $\mathbf{R}_{k,j}$, now hold the ln(**f**) values and the algorithm proceeds as follows:

Initialization:
$\mathbf{r}_{i,j} = \boldsymbol{\lambda} = \ln((1\text{-QBER})/\text{QBER})$, for all i (i=1, 2, … , N) and all j (j=1, 2, … , m(i)).

Algorithm C:

**Step1**) For each checksum, k, k=1, 2, … , M, let $\mathbf{B_k}=\{\mathbf{R}_{k,j}|i=1, 2, \dots ,t\}$ and $\mathbf{B_{k,j}= B_k\text{-}\{R_{k,j}\}}$, j=1, 2, … , t**;** then

- Compute $\mathbf{g(B_k)}$
- Compute $\mathbf{R_{k,j} = g^*(g(B_k), R_{k,j})}$ j=1, 2, … , t(k)
  Note: $\mathbf{g^*(g(B_k), R_{k,j}) = g(B_{k,j})}$

**Step2**) For every bit, i, i=1, 2, … , N

- Compute $\Lambda_i = \boldsymbol{\lambda + r_{i,1} + r_{i,2} + \dots + r_{i,m}}$ , m=m(i);
- Form Y' by flipping the bits of Y for which $\Lambda_i < 0$, otherwise use Y. Compute $D'=\mathbf{G}Y'^T$. If D'= C, the LDPC algorithm has converged; if not and provided that more iterations are allowed,
- For every i and every j=1,2, … , m(i), set $\mathbf{r_{i,j} = \Lambda_i - r_{i,j}}$
- Return to **Step1**

It should be noted that the efficient execution of this algorithm requires quick execution of the **g** and **g*** functions; quick execution nevertheless implies loss of accuracy and, as a result, there may be instances in which inequality (4) is not satisfied and, therefore, g* cannot be computed.

The requirement for quick execution can be met by using integer lookup tables. By having all logarithms represented as fractions of the form i/M with M constant (e.g., M=1024), storing i in the registers, and employing tables which map the positive i's onto $j \approx M\times\ln(1+\exp(-i/M))$ and $j' \approx M\times\ln(1-\exp(-i/M))$. The value of M implicitly determines the precision of the representations and the sizes of these tables ($\approx \ln(2M)M$) since for big u, $\ln(1\text{-}e^{-u}) \approx -e^{-u}$ and $\ln(1+e^{-u}) \approx e^{-u}$, while $Me^{-u} < 0.5$.

The requirement to avoid areas of computational instability is met by ensuring that the integers values assigned to registers are, in absolute value, no smaller than some small value (e.g. 16) and no larger than some big value (e.g., 9182).

Even so, testing revealed instances in which the evaluation of **g*** was impossible and two variants of algorithm C were tested. The cruder variant, $\mathbf{C_1}$, whenever inequality (4) was not met, would use the assignment

$$\mathbf{R_{k,j} = sign(g(B_k) * R_{k,j}) * R_{k,j}}.$$

which would leave the register's absolute value unchanged but ensure the right sign.

The other variant, $\mathbf{C_0}$, would find for each checksum, k, one register $\mathbf{R_{k,p}}$ whose absolute value is minimal among the registers associated with checksum k. It would then save $\mathbf{g(B_k - \{R_{k,p}\})}$ from the recursive computation $\mathbf{g(B_k) = g(g(B_k - \{R_{k,p}\}), R_{k,p})}$. Then for all j, j=1,2,…,t(k), if $|\mathbf{R_{k,j}}| = |\mathbf{R_{k,p}}|$, $\mathbf{R_{k,j}} = \mathbf{g(B_k - \{R_{k,j}\}) = sign(R_{k,p} * R_{k,j}) * g(B_k - \{R_{k,p}\})}$, otherwise $\mathbf{R_{k,j} = g^*(g(B_k), R_{k,j})}$.

Thus, whenever $|\mathbf{R_{k,j}}| = |\mathbf{R_{k,p}}|$, algorithm $\mathbf{C_0}$ will estimate the next value of $\mathbf{R_{k,j}}$, $\mathbf{g(B_k - \{R_{k,j}\})}$, from the already computed $\mathbf{g(B_k - \{R_{k,p}\})}$.

## 6. ALGORITHM IMPLEMENTATIONS AND RESULTS

To test the algorithms, we chose to operate on chunks of N=27,720 bits and to assume that the QBER is 4%. Given that the observed QBER is a random variable with mean of 0.04, the ability to correct up to three sigmas above the average would require at least 7159 checksums. The actual minimum should take into consideration that algorithmic efficiency requires short checksums, each of which represents less than one bit of information, that the checksum values are correlated, that actual codes are always somewhat short of perfection and that we require quick convergence. For these reasons we chose to increase the unrealistic 7159 by a hefty factor of 45% and create 10395 checksums so that each point would participate in 3 checksums and each checksum would have 8 summands (8*10395=3*27720). The matrices we used were not designed to be optimal. Our interest was centered on comparative algorithm performance which should not be affected by the choice of the matrix. Indeed, absent arithmetic errors, all three algorithms differ only in the representation of information and in the arithmetic operations they carry out. Ideally, they would be producing, step-by-step, the same numbers (albeit expressed in different scales) and results. The data, see Table 1, lists the relative performance for our three algorithms ($\mathbf{C}_0$, $\mathbf{C}_1$, **A**) and shows that **A** has slightly better performance than $\mathbf{C}_0$, both ≈9 ms per iteration, while $\mathbf{C}_1$ is about 50% slower, ≈14 ms per iteration, and requires more iterations than the other two. Algorithms $\mathbf{C}_0$ and $\mathbf{C}_1$ operate exclusively over the approximate natural logarithms of the "**f**" values. Algorithm A toggles between "**a**" values and "**f**" values.

That $\mathbf{C}_1$ may require more iterations to converge is explainable. When a checksum is computed over a set **B** of non-zero entries and |g(**B**)| fails to be smaller than all the **B** entries, we are operating near the area of formula-instability and the estimated log-likelihood values risk being grossly inaccurate.

The per iteration speed of **A** in comparison to $\mathbf{C}_0$ and $\mathbf{C}_1$ is consistent with the fact that the algorithm in **A** is simple while algorithms $\mathbf{C}_0$ and $\mathbf{C}_1$ contain more instructions and subroutine calls.

The relative advantage of $\mathbf{C}_0$ over $\mathbf{C}_1$ reflects the fact that our input (all bits equally likely to be in error initially) and our handling of the computations (artificially keeping the likelihood ratios away from 0 and their absolute values within bounds) create situations during which the inversion, i.e. the execution of Eq. (5), is not all that frequent. In one instrumented instance, the percentage of log-likelihood computations that used **g*** varied from iteration to iteration as follows:

0%, 71%, 87%, 87%, 87%, 87%, 87%, 86%, 75%, 43%, 14%, 3%, 1%, 1%

The mean of these values is 52% and given that our matrix is such that for every checksum there are 8 summands, one would expect on the average the following:

- $\mathbf{C}_0$ requires for each complete checksum computation, (8-1) executions of **g** and 8*.52=4.16 executions of **g***.

- $\mathbf{C}_1$ requires for each complete checksum computation, (8-1) executions of **g** and 8 executions of **g***.
- Given that **g** and **g*** are of similar complexity, we would expect $\mathbf{C}_1$ to require ≈14*(11/15)=10.3 ms per execution and the additional savings of 1.3 ms have not yet been understood and explained.

Table 1 shows several parameter configurations for our test sequences and the corresponding computational effort for error correction. The parameters column shows the pseudo random number generator seed, the number of bits in error (|**d**|) and the number of checksums that are initially violated. The next three columns correspond to the performance of the three algorithms tested and show the number of iterations necessary correct the errors and the time in milliseconds needed to do it.

All three algorithms were implemented in Java and run within a Java development environment. The computer used was a 4-CPU, 3.00 GHz, 1 GByte of RAM, Windows XP box and the only user application running (other than the various background tasks) was the error-correction code.

Table 1. Performance of our three LDPC implementations and their execution parameters.

| **Parameters** RNG* bit * chksum Seed * errors* errors | $\mathbf{C}_0$ iter/ms | $\mathbf{C}_1$ iter/ms | **A** iter/ms |
|---|---|---|---|
| 69/1089/2501 | 14/119 | 14/194 | 12/106 |
| 691/1110/2554 | 11/98 | 12/173 | 11/95 |
| 6912/1099/2531 | 11/97 | 13/189 | 11/95 |
| 69123/1112/2534 | 11/98 | 13/188 | 11/95 |
| 691234/1135/2533 | 12/106 | 15/216 | 13/113 |
| 6912345/1110/2556 | 10/91 | 12/170 | 10/86 |
| 691234567/1101/2491 | 10/91 | 11/166 | 10/86 |
| 91234567/1089/2507 | 11/97 | 12/170 | 11/95 |
| 12345678/1049/2419 | 9/78 | 9/123 | 8/69 |

## 7. CONCLUSION

We have presented a marginally lower limit on the entropy reductions due to QKD reconciliation and thus the reduction in secrecy of the data sequence. Contrary to what appears to be the consensus concerning LDPC algorithm efficiency [10,20], our data seem to indicate that the performance reflects not only the representation of the relevant data (such as a, f, or ln(f)), but also the data structures used and what may appear to be minor implementation details. It remains to be seen if we measured inherent properties of the algorithms or if they were affected by our parameter choices (e.g., the Gallager matrix, QBER, our generosity in the number of parity bits used to ensure rapid conversion, etc.).

## ACKNOWLEDGEMENTS

We would like to thank George Logan for a number of informative discussions and suggestions and in particular, for the idea behind algorithm $\mathbf{C}_0$. Similarly I would like to thank Daniel Genin whose ideas strengthened both the algorithms described above and their presentation.

## REFERENCES

[1] Bennet, C. H. and Brassard, G., "Quantum Cryptography: Public key distribution and coin tossing", Proc of the IEEE Intern'l Conf on Computers, Systems, and Signal Processing, Bangalore (1984).

[2] Brassard, G. and Salvail, L., "Secret-Key Reconciliation by Public Discussion", Proc of Eurocrypt'94, Lecture Notes in Computer Science, 765, Spriger Verlag, 410-423 (1994).

[3] Bennet, C. H., Brassard, G., and Robert, J., "Privacy Amplification by Public Discussion", SIAM J. COMPUT., Vol 17, No. 2, (Apr 1988).

[4] Bennett, C. H. and others, "Experimental Quantum Cryptography", J. of Cryptology 5, (1992).

[5] Bennet, C. H. and others, "Generalized Privacy Amplification", IEEE Transactions on Information Theory, Vol. 41, No. 6, pp 1915-1923, (Nov 1995).

[6] Branciard, C., Gisin, N., Kraus, B. and Scarani, V., "Security of two quantum cryptography protocols using the same four qubit states", Physical Review A 72, 032301 (2005)

[7] C. Cachin, "Entropy Measures and Unconditional Security in Cryptography", Ph.D. thesis, ETH Zurich (1997)

[8] Digital video broadcasting (DVB); "second generation framing structure, channel coding and modulation systems for broadcasting, interactive services, news gathering and other broad-band satellite applications", EN 302 307 V1. 1.1, European Telecommunications Standards Institute (ETSI), (2006).

[9] Elliott, C., Colvin, A., Pearson, D., Pikalo, O., Schlafer, J. and Yeh, H., "Current status of the Darpa Quantum Network", (Mar 2005). <http://arxiv.org/ftp/quant-ph/papers/0503/0503058.pdf>

[10] Eroz, M., Sun, F. and Lee, L., "DVB-S2 low density parity check codes with near Shannon limit performance", Intern'l. Journal Satell. Comm. Networks. 22:269-279, (2004).

[11] Gallager, R. G., "Low-density parity-check codes", No. 21, MIT Research Monograph Series, MIT press, (1963), IRE Transactions on Information Theory, 8(1):21–28, (Jan 1962).

[12] Falcao, G. and others, "How GPUs Can Outperform ASICs for Fast LDPC Decoding", *Proc of the 23rd Intern'l Conf on Supercomputing,* Session on Accelerating Applications with GPUs, pp 390-399, (2009)

[13] Hughes, R. J., Nordholt, J., Derkacs, D..and Peterson, C, "Practical free-space quantum key distribution over 10km in daylight and at night", New Journal of Physics 4 43 (2002). <http://iopscience.iop.org/1367-2630/4/1/343>

[14] Lo, H. K., Ma, X. and Chen, K., "Decoy State Quantum Key Distribution", Physical Review Ltrs 94, 230504, (Jun 2005).

[15] Mackay, D. J., "Information Theory, Inference, and Learning Algorithms", Cambridge University Press, (2003).

[16] Mink, A. Tang, X., Ma, L., Nakassis, T., Hershman, B., Bienfang, J., Su, D., Boisvert, R., Clark, C. and Williams, C., "High Speed Quantum Key Distribution System Supports One-Time Pad Encryption of Real-Time Video," Proc of SPIE Defense & Security Symposium, Orlando, FL, 17-21, 6244, 62440M1-7, (Apr 2006).

[17] Myers, J. M., Wu, T. and Pearson, D., "Entropy estimates for individual attacks on the BB84 protocol for quantum key distribution", Proc SPIE, vol. 5436, pp 36-47, (Apr 2004).

[18] Nakassis, A., Bienfang, J. C., Johnson, P., Mink, A., Rogers, D., Tang, X. and Williams, C. J., "Has Quantum Cryptography been proven Secure?", Proc SPIE, Vol. 6244, pp 62440I-1-62440I-9 (Apr 2006).

[19] Pearson, D., "High-speed QKD Reconciliation using Forward Error Correction", Proc. 7th Intern'l Conf on Quantum Communication, Measurement, and Computing (QCMC), pp 299-302, (2004).

[20] Pirou, F., "Introduction of Low-Density Parity-Check Decoding Algorithm Design", Master Thesis Linköping University, (2004). <http://liu.diva-portal.org/smash/record.jsf?pid=diva2:19490>

[21] Renner, R., Gisin, N. and Kraus, B., **"**An information-theoretic proof for QKD protocols"**,** Physical Review Ltrs A, American Physical Society, Vol. 72, No. 012332, (Jul 2005). <http://arxiv.org/abs/quant-ph/0502064>

[22] Scarani, V., Bechmann-Pasquinucc, H., Cerf, N., Dusek, M., Lutkenhaus, N. and Peev, M., "The Security of Practical Quantum Key Distribution", (Oct 2010). <http://arxiv.org/PS_cache/arxiv/pdf/0802/0802.4155v2.pdf>

[23] Shokrollahi, A., "LDPC Codes: An Introduction", (Apr 2003). <http://www.telecom.tuc.gr/~alex/papers/amin.pdf>

[24] Slutsky, B., Rao, R., Sun, P. and Fainman, Y., "Security of quantum cryptography against individual attacks", Physical Review A, Vol. 57, No. 4, pp 2383-2398, (Apr 1998).

[25] Slutsky, B. ., Rao, R., Sun, P., Tancevski, L. and Fainman, Y.,, "Defense frontier analysis of quantum cryptographic systems", Applied Optics, Vol. 37, No. 14, pp 2869-2878, (May 1998).

[26] Van Asche, G., "Quantum Cryptography and Secret-Key Distillation", Cambridge University Press, (2006).