Expeditious Reconciliation for Practical Quantum Key Distribution

Anastase Nakassis[1], Joshua Bienfang, Carl Williams[2]
NIST, 100 Bureau Drive, Gaithersburg, MD 20899-8423

Abstract

The paper proposes algorithmic and environmental modifications to the extant reconciliation algorithms within the BB84 protocol so as to speed up reconciliation and privacy amplification. These algorithms have been known to be a performance bottleneck [1] and can process data at rates that are six times slower than the quantum channel they serve[2]. As improvements in single-photon sources and detectors are expected to improve the quantum channel throughput by two or three orders of magnitude, it becomes imperative to improve the performance of the classical software. We developed a Cascade-like algorithm that relies on a symmetric formulation of the problem, error estimation through the segmentation process, outright elimination of segments with many errors, Forward Error Correction, recognition of the distinct data subpopulations that emerge as the algorithm runs, ability to operate on massive amounts of data (of the order of 1 Mbit), and a few other minor improvements. The data from the experimental algorithm we developed show that by operating on massive arrays of data we can improve software performance by better than three orders of magnitude while retaining nearly as many bits (typically more than 90%) as the algorithms that were designed for optimal bit retention.

keywords: Reconciliation, Cascade, Forward Error Correction, Quantum Key Distribution, Quantum Cryptography

1. Introduction

On-line cryptography relies on transforms that use a shared secret to transform the communicated plaintext. As the ongoing use of the shared secret, known as a traffic key, compromises its secrecy, new traffic keys are created through public negotiation that relies on mathematical transforms that are easy to compute and extremely difficult to reverse (i.e. some function f such that f(x) is easy to compute and the equation f(x)=a is, for practical purposes, impossible to solve). Therefore the secrecy of the communicated information can be compromised any time in the future should the (extremely well) educated guesses on the complexity of some computations prove to be in error.

An alternative scheme for creating shared secrets, known as BB84[2], was suggested about 20 years ago and is based on quantum communications that would generate, so to speak, perfect cryptographic protection provided that our understanding of the laws of nature does not undergo radical change. Unfortunately, the scheme had and still has limitations and engineering problems that render it rather impractical. Paradoxically, the early implementations of BB84[1,5] show that the classical algorithms within the BB84 protocol can be performance bottlenecks. Specifically, through quantum communications BB84 creates in hosts X and Y bit-strings s[x] and s[y] that in the absence of noise and eavesdropping are identical. As neither of these assumptions is necessarily true, X and Y expect the received string, s[y], to be of the form s[x]⊕**d** and in the absence of any other information are willing to attribute all non zero terms of **d** to eavesdropping. A classical algorithm is used to correct, with great probability, all errors in s[y] while releasing minimal information to potential eavesdroppers. Once this algorithm (Reconciliation) has ended, the strings at X and Y are transformed into shorter strings in a process (privacy amplification) that eliminates (with great probability) whatever information eavesdroppers may have collected during the quantum communication phase and Reconciliation.

---

[1] anakassis@nist.gov; phone 1 301 975-3632; fax 1 301 590-0932; nist.gov
[2] This work was supported, in part, by the DARPA QuIST program

We note that the quantum communication scheme used is such that when an eavesdropper, by convention Eve, tampers with some bit b in s[x], there is a 0.5 probability that Eve will read the bit without detection and a 0.5 probability that the bit read by Eve and Y will be for all intents and purposes a random variable that conveys no information whatsoever on the value X encoded. Thus, each tampered bit appears as 1 in **d** above with probability 0.25. The protocol assumes that Eve is trying to collect information about the strings s[x] and s[y], not to mount a denial of service attack and consequently concerns itself with the problem of correcting s[y] when relatively few bits have been tampered with. As a rule of thumb, if more than 60% of the bits have been tampered with, which roughly translates that more than 15% of the bits in **d** equal 1, the data will be thrown away. The reader should note that the percentages cited are actually the expected values of random variables and that knowledge of the frequency of errors can only translate into confidence intervals for the frequency of bit tampering.

The reconciliation and privacy amplification algorithms are generally run as follows:

1 A random subset of the measured bits is exchanged to estimate the percentage, p*, of the 1's in **d**. Randomness is essential so that Eve not be capable of tampering without detection.
2 p* is used as an optimizing parameter in an interactive algorithm that attempts to detect and correct errors in s[y]. The parameter p* is also used to determine, with very high probability, that all errors have been corrected. Interactivity ensures parsimonious use of bits.
3 A random hashing function f is communicated between X and Y so as to map s[x] and the presumably corrected s[y] onto identical strings in which all information pirated by Eve during the quantum communication and step 2 above is eliminated with very high probability.

For details on steps 1-3, the reader may consult [3,4,7,8]. For the purposes of this study we shall treat the algorithm described in the Brassard-Salvail paper[3] as normative.

It has of course not escaped the attention of previous authors[8] that

- Step 1 can be eliminated if step 2 can be run without reference to p*. When step 2 is completed, the communicating parties know, depending on the algorithm, either the number of 1's in d or a probabilistic bound.
- Most of the interactivity in step 2 can be dispensed with if we can be, so to speak, not parsimonious with the number of bits we are willing to sacrifice and prepared to deal with the effects of errors introduced during "error-correction"

But, it also appears that the problem has become harder because it has been conceived too narrowly:

- Communicating Systems can be expected to use the past to derive a rough initial estimate p*.

- The language of error correction has led researchers to the perception that s[y] must be transformed into s[x]. Nevertheless, neither of these strings is more authentic or more random than the other and once the quantum communication has taken place, the situation between sender and receiver is entirely symmetric. Architectures and algorithms that explicitly acknowledge this symmetry are likely to be better performing than their asymmetric brethren. In particular, there can be advantages in visualizing the algorithm as a random process that tends to reduce the discrepancies between s[x] and s[y] by transforming, and possibly reducing in length, both of them towards some common value s[x,y].

In the sections that follow we will flesh out these observations and will put forth an algorithm variant whose performance will be compared to Cascade's.

2. **An overview of Cascade.**

Before we attempt to see if plausible alternatives to Cascade exist, let us try to get a better idea of how this algorithm operates, with emphasis at the opening stages that determine the overall performance because over 50% of the bits in error will be corrected during the first pass.

[Cascade, as described in reference[3]]

1. Release enough bits to obtain a "good" estimate of p, p*
2. Use p* to optimally segment what remains of s[x] and s[y].
3. For each segment compute and compare the checksum.
4. Whenever the checksums differ (odd number of errors present), iteratively and interactively divide the segment (choose each time the halves that differ in checksum values) till a single error is located and fixed.
5. Shuffle the bits and repeat steps 2, 3, and 4 for the resulting segments. If the corrections disturbed previously matching checksums, iteratively and interactively locate and correct the corresponding errors until all corresponding checksums match.
6. Repeat step 5 until satisfied that there are no remaining errors.
7. Randomly choose a hash function and map the reconciled strings to shorter strings in a way that Eve retains no useful information with an extremely high probability.

The reader should note that the above summary is at times intentionally vague. For example, many authors suggest that half of the bits should be used to estimate p and my understanding is that they do so because either they are not very sensitive to the practical aspects of the algorithm or because they have been conditioned to think of an algorithm that operates over a few hundred bits (the published examples[1,5] refer to instances of 200+ up to 700+ bits). At the other end, those who care about performance can cut the Gordian knot by assuming that p*=0.15 or thereabouts and have the algorithm abort, if the actual p is even bigger. As these are matters of optimization within specific algorithms and environments, the less we say on the subject, the better.

We note that Reference[3] includes a list of optimal segment lengths. For p in {0.01,0.05,0.10, 0.15} the corresponding lengths n[p] are {73,14,7,5}.

We observe that p*n[p] varies between 0.7 and 0.75, a value that is close to ln(2)= 0.69, and that for long segments the number of errors in a segment is approximately Poisson with parameter λ=ln(2). Back of the envelop calculations indicate that roughly 50% of the blocks contain no error, 35% one, 13% 2, and 4% 3 or more. Indeed, if we compute the probabilities from the Bernoulli distribution we find:

Errors	p=0.01, n=73	p=0.05, n=14	p=0.10, n=7	p=0.15, n=5
0	0.48	0.49	0.48	0.44
1	0.35	0.36	0.37	0.39
2	0.13	0.12	0.12	0.14
3	0.03	0.03	0.02	0.02
4	0.005	0.004	0.002	0.0002
5	0.0007	0.0004	0.0002	0.00008
6	0.00009	0.00003	0.000006	0.0
7	0.000008	0.000002	0.0000001	0.0
8 or more	0.00000005	0.0000004	0.0000002	0.0

The data also show that about 60% of the segments have an even number of errors. Indeed, if P[m] is the probability that a segment of length m contains an even number of errors,

$$P[0]=1,\ P[1]=1-p$$
$$P[m+1]=p(1-P[m])+(1-p)P[m]$$

whose solution is $P[m]=0.5*(1+(1-2p)^m)$.

Back of the envelop computations show that if pn~ln(2), P[n]~0.5(1+1/4)=0.625. Indeed for the (p,n) values in the table above we obtain {P[73]=0.61, P[14]=0.61,P[7]=0.60, P[5]=0.58}.

These results suggest that one can bypass the estimation of the error rate, p, by initially dividing the bitstring in segments of length N[0] and exchanging the parities of these segments. Unless the parities agree for an acceptable frequency, we recursively split the segments at hand until we either find an acceptable partition or failure is diagnosed. We note that:

- When we stop, the sum total of the information released is the information released dutring the last step.
- This technique lacks statistical significance unless we are processing long strings.

The Cascade algorithm uses a subroutine, **binary**, to ferret out errors when the parities of two sets differ. The idea is quite simple. We divide the original set into two roughly equal parts (cardinality-wise) and exchange the parity of the first half (hence, implicitly, the parity of the second). Through successive splittings, we will eventually isolate a bit in error. The question is how much information we reveal, on the average, when we run **binary** on a set of length n. To simplify the computation, let us assume that the bitstrings in question differ only in one bit location.

We note that if I[n] is the information revealed on the average, then

$$I[1]=0,$$
$$I[2n]=1+I[n],$$
$$I[2n+1]=1+(n/(2n+1))I[n]+((n+1)/(2n+1))I[n+1]$$

whose solution is $I[m]= k+2x/m$ with $m=2^k+x$ and $0\leq x<2^k$.

For instance, if m=73, I[m]=6+18/73~6.25.

We observe that this value is close to $\log_2(m)$ (e.g., $\log_2(73)=6.19$) and that the algorithm is, indeed, near optimal. We also observe that if the algorithm operates on several segments in parallel, the steps needed for all errors to be located is almost always k+1 so that if m=73 (or any number between 65 and 128 included, we would need 7 steps before errors were isolated in all segment

On the other hand a Hamming code[6] of 7 bits adjusted for the circumstances at hand would immediately correct all single bit error at the cost of possibly introducing new errors if the total bits in error were 3 or more. This, as we will see, represents a tractable cost.

Finally, it is worth investigating the effects of choosing the length of the initial segments too small. If , for example, p=0.01 and we operate on 40 bit long segments we shall correct at the first pass 70% of the errors and sacrifice 6.3% of our bits. If we choose the optimal 73 bit length segments we shall correct 53% of the errors while sacrificing 4.7% of the bits. If p=0.05, for 10 bit segments the corresponding percentages are 66% and 21% while for the optimal 14 bit segments they would have been 56% and 18%.

These numbers suggest that be time improvements can be achieved at low bit cost by purposefully underestimating the segment lengths in the first partition and possibly beyond.

3. **An alternative to Cascade**

Bearing the above remarks in mind we implemented a tested a Cascade variant which has four steps:

1. **Initialization:** find a workable starting segmentation of the strings to be reconciled
2. **Reconciliation:** Remove (presumably all) inconsistencies between the two strings
3. **Reconciliation check**
4. **Privacy amplification.**

Our alternative is based on the idea that while for practical purposes the computational power is infinite, both bits and time delays (cost of communications) matter. It follows then that we are led to different views of what must be preserved and what jettisoned. In summary the algorithm we propose would run as follows:

3.1 Initialization

The bits are shuffled, and parities are exchanged for some reasonable segment length. If the observed parity distribution meets our needs, we have a starting configuration and an estimate for the number of altered bits. If not, a new (shorter) length is chosen and the step is repeated. If no workable length emerges, we abort.

3.2 Reconciliation and reconciliation check.

Cascade differs from previous reconciliation algorithms in that it does not discard bits to eliminate on the spot whatever information is released through negotiation. This has the advantage of maintaining control over the parities of the original segments. As a result, whenever an error is found in a segment after the first pass, matching errors in previous passes are exposed and corrected. On the other hand, this technique will also lead us to correct bits that will eventually be discarded. This does not seem to be much of a problem at low error rates but it might become one when the error is high. For example, if we are working with p=0.15 and 5 bit segments, it takes one bit to indicate odd parity and 2.4 bits to correct a single error. If the segment holds a single error the eavesdropper's expected knowledge on the 4 non error bits is on the average at least 1.41 bits [(5-1)*(2p)/(1-p), p=0.15], the information that he could have collected by guessing databases and retransmitting the measured values]. So that the net average yield will be less than 5-3.4-1.41=0.19 bits. If we now consider that, according to our table, there is a probability of about 0.02/0.41 or, roughly, 0.05 that a segment with an odd number of errors has more than one, we see that some corrections are counterproductive in bits as well as in time.

We observe that once parities have been exchanged, the segments are naturally divided into two subpopulations. Those that have matching parities for which the effective error rate is about 3/5 the error rate in the general population, p, and those with differing parities that give as segments for which the effective error rate is about 5p/3. This remark has led us to an algorithm that operates as follows:

- Short segments with low or negative yield, e.g., length=5 and odd errors are discarded
- Error correction code is used to correct when an odd number of "errors" are present
- Impossible corrections (e.g., the Error Correction code demands that bit #12 of a ten bit segment be flipped) indicate 3+ corrections and the segment is discarded.
- At each step we maintain up to two populations (a-bits and b-bits) with different profiles. The basic algorithm is run separately on each subpopulation and the four resulting subsequences (non-corrected a and b segments and corrected a and b segments) are recombined into two.
- At the end of each step we suppress bits in each segment so that Eve's information does not increase.

When the collected observations indicate an overall remaining error probability smaller than 10^{-4}, we combine the two populations, divide the total into 100 bit segments, and exchange 7-bit Hamming codes. When the codes differ, the whole segment is discarded. This ensures that a segment with errors will fail to be discarded with probability less than 10^{-8}.

The process we just described has two disadvantages:

- It is bitwise suboptimal, and
- It provides incomplete and indirect information as to the number of errors in the original string. This, in turn, requires accounting of the effects of discarding bits. As a result, the upper bound of the estimates on the information that Eve possesses is more generous than in Cascade.

3.3 Reconciliation check

As in all algorithms, the negotiating parties compute and exchange a checksum. If the checksums differ, the negotiation aborts. In our case the checksum is 60 bits long, thereby ensuring that the probability that we will fail to abort when we should is less than 2^{-60} or, roughly, 10^{-18}

3.4 Privacy amplification

The faulty corrections and the wholesale dumping of segments with too many errors during Reconciliation make the exact number of errors present in the original string unknowable. But, what we really need to know is an upper bound U(v) such that the probability that Eve's information on the final string exceeds U(v) is v or less. Clearly, such bounds can be derived although they will be less tight than those of Cascade because of the additional uncertainty the algorithm introduces.

4. Comparative results

We implemented the ideas above in a prototype and the initial results were better than expected. Indeed, examples of Cascade found at the university of Aarhus[1], Denmark, http://www.cki.au.dk/experiment/qrypto/templates/initiator/RecBB84/output2_1.html, can be summarized as follows:

Original configuration	Round trips	Final result
Bits=233 bits, 9 errors estimated p=0.02 estimation used 48 bits	~ 24-28 bits revealed 57+10	233-48=185 bits corrected bits Upper bound on Eve's knowledge=57+10+26=93 bits Secret= 92 bits, percent= 0.39
Bits=259, 15 errors estimated p=0.02, estimation used 48 bits	~31-36 bits revealed 99+10	259-48=211 corrected bits Upper bound on Eve's knowledge= 99+10+39=148 Secret=63 bits, percent= 0.24
Bits=746, 21 errors estimated p=0.0333.., estimation used 136 bits	~ 40-46 bits revealed 128+10	746-136=610 corrected bits Upper bound on Eve's knowledge= 128+10+52=190 Secret= 420 bits, percent= 0.56

We note that:

- The number of roundtrips (not cited) was inferred by the data and cited as an interval (lower bound, upper bound). The lower bound has very small probability of ever being achieved. The upper bound is very close to the average case when a string is partitioned in many segments. This was not the case here
- The algorithm's bit-retention percentages reflect, in part, the shortness of the strings used.
- The extra summand in Eve's knowledge represents a probabilistic upper bound (three sigma) on the number of the quantum bits Eve managed to read successfully.

The data we collected are not directly comparable to the ones we reference but suggest that we have significantly improved on the performance of the cited software and we believe that further improvements are possible.

The experimental Java software we developed was tested between two modest profile PCs (1.8 GH/1Gb and 1.0GHz/128Mb) linked over a 100Mbs Ethernet. In the absence of interference over the Ethernet, the software processes 1.3 million bits (of which 4% are in error) in less than three seconds and sacrifices 45% of the bits (against a theoretical minimum of 34%). It is not as fast as desired but algorithmic improvements, compiled code, and improved hardware configurations should make it as fast as the hardware. It also uses a more generous upper bound than the one we applied above as it assumes that Eve will in the end know which bits were in error (mathematically untrue for our algorithm but from an engineering point acceptable) and that Eve is using the Breidbart basis to measure and resend photons[5].

In the first experiment we hold the error rate fixed at 0.04 while varying the number of bits to be processed and the number of modules that executed the algorithm. Each module executed the algorithm five times so as to produce more representative data and the supervising software measured in milliseconds the time from the moment the first module was launched until the last concluded its task. The numbers we collected are:

	12 Kbits	120 Kbits	600 Kbits	1.2 Mbits
1 module	15467ms	14061 ms	12906 ms	26373 ms
2 modules	16749ms	14655 ms	17218 ms	30702 ms
3 modules	17390ms	16390 ms	21045 ms	42200 ms
4 modules	17234ms	16530 ms	24967 ms	53106 ms
5 modules	17061ms	16577 ms	29873 ms	64996 ms
6 modules	18045ms	17467 ms	34904 ms	76964 ms
7 modules	18140ms	17327 ms	41045 ms	89479 ms
Typical output length (reconciles and amplified	~6K	~63K	~336K	~700K

The data are initially hard to accept (it takes less time to process 120K bits than it takes to process 12K) and while we can theorize why this is so, we are not sure. Experiments also show that there can be great variability in the measurements, which reflects the network status. The important thing is that when the computer does not run other tasks, the sweet spot for our configuration seems to be around four concurrent negotiations handling 600K each of them requiring, roughly, 1.25 seconds (24967/20).

The second experiment carried varied the probability of error in order to show the time the algorithm needs to converge, the reconciled bits produced, and the size of the shared secret. As the table below shows, there is a big jump around 0.05 (at least in part because of the segmentation algorithm).

Error frequency	Max bits recoverable	Three sigma bound on Eve's information	Actual bits recovered (our variant)	Computed three sigma bound on Eve's information	Shared secret size	Time
0.010	1157K	31K	1087K	32K	1055K	2.3s
0.020	1080K	62K	992K	64K	928K	2.7s
0.030	1014K	93K	892K	98K	794K	2.7s
0.040	953K	124K	834K	127K	707K	2.5s
0.050	898K	154K	774K	160K	614K	4.8s
0.060	846K	185K	585K	153K	432K	5.3s
0.070	798K	216K	539K	173K	366K	4.7s
0.080	752K	247K	520K	192K	328K	4.5s
0.090	709K	277K	474K	212K	262K	4.4s
0.100	668K	308K	405K	230K	175K	4.7s
0.110	629K	339K	385K	247K	138K	4.9s
0.120	592K	369K	321K	265K	56K	4.9s

The anomaly that appears at 6% and beyond is an artifact of our decision to throw away all five bit segments containing errors. As these segments represent about 40% of the data we end up working with the remaining 60%. The drop in the computed information for Eve reflects the massive data discarding of the algorithm

5. **Conclusion**

We would like to caution the reader that

- The times cited above have a great degree of variability as there were background processes that compete for resources
- As Cascade was run on short strings allowances must be made for the fact that fixed (or near fixed) costs loom large.
- Direct time comparisons between algorithms that operate on different environments are not meaningful.

Nevertheless we believe that the data at hand demonstrate that:

- reconciliation operating on big strings and using FEC cuts down on the instances of interaction and achieves dramatically improved results;
- in parallel module execution balances the resource use among modules and maximizes resource use;
- alternatives to Cascade exist that have similar bit retention profiles, less algorithmic complexity, and superior overall performance (measured in secret_bits/second).

It should also be noted that the software algorithm implementation is not fully optimized and that standard IT techniques could improve throughput by a factor between 2 and 10.

References

1. An excellent Web site to explain BB84, Reconciliation and Cascade at http://www.cki.au.dk/experiment/qrypto/doc/QuCrypt/bb84prot.html
2. C.H. Bennet and G. Brassard, " Quantum cryptography: Public key distribution and coin tossing", *Proc. of the IEEE, Int. Conf. on Computers, Systems, and Signal Processing*, 175-179 (conf. held in Bangalore, India, December 1984).
3. Gilles Brassard and Louis Salvail, "Secret-Key Reconciliation by Public Discussion", *proceedings of Eurocrypt'94, Lecture notes in computer Science*, 765, Spriger Verlag, 410-423.
4. Christian Cachin, Ueli M. Maurer, "Linking Information Reconciliation and Privacy Amplification", *journal of Cryptology* **10(2)**: 97-110 (1997)
5. Richard J Hughes, Jane E Nordholt, Derek Derkacs and Charles G Peterson,"Practical free-space quantum key distribution over 10 km in daylight and at night", *New J. Phys.* **4** (2002) 43.
6. Shu Lin and Daniel J. Costello, *Error Control Coding*, 79-82 and 111-116,Prentice Hall, Englewood Cliffs, 1983.
7. M.A. Nielsen and I.L.Chuang, *Quantum Computation and Quantum Information*, 586-603, Cambridge University Press, Cambridge, 2000.
8. Peter M. Nielsen and others, "Experimental Quantum Key Distribution with Proven Security Against Realistic Attacks", *Journal of Modern Optics*, vol **48**, no. 13, 1921-1942, 2001.