

Probabilistic Erasure-Resilient Codes

Jeff Edmonds*

Michael Luby†

1 Introduction

Defⁿ: An **erasure-resilient code** is specified by a triple (m, n, ℓ) . It is a function E that maps messages $M = (M_1, \dots, M_m)$ consisting of m blocks onto encodings $E(M) = (E_1(M), \dots, E_n(M))$ consisting of n blocks, in such a way that any m of the blocks $E_{i_1}(M), \dots, E_{i_m}(M)$ of $E(M)$ together with their indices i_j uniquely determine the message M . Here the blocks are bit-strings consisting of ℓ bits and $\ell' = \ell/32$ or $\ell/64$ computer words. This code is often referred to in the literature as a **?? minimum distance code**.

Erasure-resilient codes are resilient against the code blocks being lost, but not against a block being corrupted. It assumes that the indices of the lost blocks are known. These are different from those of error-correcting codes, where the individual bits can be corrupted and the locations of corrupted bits are not known in advance. The assumptions for the erasure-resilient codes are realistic for applications like robust data transfer on packet-based networks [?], information dispersal [?], or secret sharing [?].

The asymptotically fastest known algorithm for erasure-resilient codes have encoding and decoding complexities of $\mathcal{O}(n \log^2 m)$ and $\mathcal{O}(m \log^2 m)$ division operations over the finite field $\text{GF}[2^\ell]$, where ℓ is $\mathcal{O}(\log m)$. Divisions over $\text{GF}[2^\ell]$ are costly operations and as well the hidden constant is large. For large messages, this may be not be practical. In order to save on computation time, this paper defines probabilistic erasure-resilient codes and gives an efficient encoding and decoding algorithm.

Defⁿ: A **probabilistic erasure-resilient code** is specified by five parameters $(m, n, \ell, \epsilon, \delta)$. It is the same as a deterministic erasure-resilient code, except that it is probabilistic in the following sense. Both the encoding and the decoding have access to the same sequence of random bits. Then the scheme guarantees that any $(1 + \epsilon)m$ of the code blocks $E_{i_1}(M), \dots, E_{i_{(1+\epsilon)m}}(M)$ together with the indices i_j determine the message M with probability $1 - \delta$, where the probability is over the random bits.

The encoding and decoding times of the algorithms presented are $(6.8 + 53\epsilon)\frac{\ln(m/\delta)}{\epsilon}\ell'n$ and $(6.8 + 53\epsilon)\frac{\ln(m/\delta)}{\epsilon}\ell'm$ bit-wise exclusive-ors, where the number of computer words per block ℓ' is at least $1.2 \log(\ln(m/\delta)/\epsilon^2) \ln(m/\delta)$. This is fewer operations than for known deterministic codes. Note that the encoding time per word of the encoding is only $\mathcal{O}\left(\frac{\ln(m/\delta)}{\epsilon}\right)$ exclusive-ors rather than $\mathcal{O}\left(\ln^2(m)\right)$ divisions over $\text{GF}[2^\ell]$ and the decoding time is the same, but per word of the message.

*Supported by an *NSF* postdoctoral fellowship and by a Canadian *NSERC* postdoctoral fellowship.

†Partially supported by *NSF* operating grant CCR-9304722, *Israeli-U.S. NSF BiNational Science Foundation* grant No. 92-00226 and *ESPRIT BR Grant EC-US 030*.

For large messages and for reasonable sized constants ϵ and δ , this is a notable improvement. The bound on ℓ' is only our recommendation and is set so that the fixed costs of inverting a certain matrix is amortized by the length of the block. It is not an unreasonable block length, because there is a lower bound that states that deterministic codes require $\ell \geq \log m$.

Reed-Solomon-codes are the deterministic erasure-resilient codes that are implemented in practice. However, we describe only Interpolation-Codes because they are simpler. In these codes, the m blocks of the message are viewed as the coefficients of a polynomial M of degree $m - 1$ over the finite field $\text{GF}[2^\ell]$. The n code blocks are simply this polynomial evaluated at n distinct places. Hence, from any m of the code blocks, the polynomial M can be recovered.

Using fast Fourier transformations to manipulate the polynomials, Reed-Solomon-codes can be encoded and decoded using only $\mathcal{O}(n \log^2 m)$ and $\mathcal{O}(m \log^2 m)$ divisions over the field $\text{GF}[2^\ell]$. These operations, however, are computationally expensive and the hidden constant is large. ??? states that fast Fourier transformations are only useful for multiplying polynomials when the sum of their degrees is at least 600. In general, the following more brute force method of encoding and decoding is used.

Both Reed-Solomon-codes and Interpolation-Codes are examples of *linear erasure-resilient codes*. Such a code is described by an $(n \times m)$ matrix V over the field $\text{GF}[2^\ell]$. The message is viewed as the vector M of length m with entries being the blocks M_j viewed as elements of the field $\text{GF}[2^\ell]$. Similarly, the encoding is viewed as the vector E of length n . The linear encoding scheme then is given by $V \times M = E$. Encoding consists simply of matrix multiplication. To decode, let \hat{V} and \hat{E} be the rows of the matrices V and E corresponding to those code blocks that were received. The relation $\hat{V} \times M = \hat{E}$ still holds and the message M can be recovered as long as the matrix \hat{V} has rank m .

An erasure-resilient code is called a *systematic code*, if the first m blocks of the encoding is the message M itself. A linear code is a systematic code if the first m rows of its generating matrix V form the identity matrix. It is not hard to see that encoding a systematic code over $\text{GF}[2^\ell]$ requires $\mathcal{O}(m(n - m))$ multiplications in $\text{GF}[2^\ell]$. When decoding suppose that amongst the m code blocks received, $m - k$ of them are message blocks and k of them are redundant blocks. Then $\mathcal{O}((m - k)k)$ arithmetic multiplications are required to multiply the $m - k$ known message blocks by the matrix V and subtract the result from the redundant blocks. What remains of V is a $(k \times k)$ -matrix. Normally, inverted such a matrix requires $\mathcal{O}(k^3)$ multiplications, however, for Reed-Solomon-codes the matrix is of a special form so that it can be inverted in $\mathcal{O}(k^2)$ time. Finally, the k missing message blocks can be recover with $\mathcal{O}(k^2)$ more multiplications. These time are reasonable when the amount of redundancy $n - m$ is small (note $k \leq n - m$). However, we will be considering the scenario when $n - m$ is $\mathcal{O}(m)$. In this case, encoding and decoding require $\mathcal{O}(m^2)$ multiplications. For large messages, this may prove to be impractical.

2 Random Boolean Matrices

The high computational cost of field operations over $\text{GF}[2^\ell]$ was our first motivation for this paper. One way to avoid this is to work over the field $\text{GF}[2]$. Any $(n \times m)$ -matrix V over $\text{GF}[2^\ell]$ can be converted into a $(n\ell \times m\ell)$ -matrix over $\text{GF}[2]$ by mapping each field element in $\text{GF}[2^\ell]$ to a $(\ell \times \ell)$ boolean matrix. This significantly speeds up the computation time.

An even more ideal situation would be if the linear code itself could be over $\text{GF}[2]$. Such a

code would be described by an $(n \times m)$ matrix V over $\text{GF}[2]$. Each block M_j of the message would be viewed as a row of ℓ field elements from $\text{GF}[2]$ and the entire message would be viewed as the $(m \times \ell)$ -matrix M . Similarly, the encoding would be viewed as the $(n \times \ell)$ -matrix with rows given by blocks $E_i(M)$ of the encoding. As before, the encoding scheme is given by $V \times M = E$. A code block E_i is simply the bit-wise exclusive-or of the message blocks specified by the vector V_i , i.e.

$E_i = \bigoplus_{V_{(i,j)}} M_j$. The task of adding two boolean vectors of length ℓ can be done in “parallel” using only $\ell' = \ell/32$ or $\ell/64$ computer operations depending on the computer’s word length. The time is particularly fast, because the exclusive-or is one of the operations that computers do fast.

It is provable, however, that such a boolean deterministic erasure-resilient code does not exist. On the other hand, the following lemma proves that a random matrix V is in fact a probabilistic erasure-resilient code.

Lemma 1 *Consider a random $(m+c \times m)$ -boolean matrix, where each entry is chosen independently from $\{0, 1\}$ with probability $\frac{1}{2}$. The probability that this matrix has full rank is at least $1 - 2 \cdot e^{-0.1c}$*

Proof of Lemma 1: Consider the i^{th} equation. The probability that it is linearly dependent on the previous equations is the probability that a random vector from the m dimensional space lands in the $i - 1$ dimensional sub-space spanned by the $i - 1$ previous equations. This probability is $2^{-(m-i+1)}$. Consider the first $m - 0.28c$ equations. The probability that they are dependent is at most $\sum_{i \in [1..m-0.28c]} 2^{-(m-i+1)} \leq 2^{-0.28c} = e^{-0.1c}$. Consider one of the remaining $0.28c + 1c$ equations. If the equations before it do not have full rank, then the probability that it increases the rank is at least $\frac{1}{2}$. Hence, the choosing these $1.28c$ equations can be thought of as $1.28c$ bernoulli trials. The matrix has full rank if at least $0.28c$ of the trials succeed. The expected number of successes is $0.64c$. The probability of getting fewer than $0.28c$ is at most $e^{-\frac{(0.36c)^2}{2 \cdot 0.64c}} = e^{-0.1c}$. ■

Note that probabilistic erasure-resilient codes allow for a linear number of extra code blocks (ϵm), but that these require only a constant number $10 \ln(1/\delta)$.

3 Bundles

The linear erasure-resilient codes using matrix multiplication require $\mathcal{O}(m)$ operations per block of the message. For large messages, this may prove to be impractical. A simple solution (and one used in practice) is to break the message into fixed size pieces, called *bundles*, of b blocks each. Each of the m/b bundles is separately encoded into bn/m code blocks each, for a total of n code blocks. The number of operations per block of the message for this scheme will be $\mathcal{O}(b)$. If the bundle size b is small, then this is a marked improvement.

The problem with this scheme is that one could receive m of the n code blocks, without receiving b about each of the bundles. In this case, the entire message would not be recovered. What is done in practice is simply to hope that enough code blocks are received about each bundle. This can be formalize as follows. Randomly permute the code blocks. Then if any $(1 + \epsilon)m$ of the code blocks are received, the expected number of blocks received about a particular bundle is $(1 + \epsilon)b$. The goal is to bound the probability that enough code blocks are received about each bundle. This motivated defining a probabilistic erasure-resilient code.

The following analogy is useful. Consider $\frac{m}{b}$ buckets (bundles) of size b and $(1 + \epsilon)m$ balls (code blocks) thrown randomly at the buckets. The expected number of balls to land in a single bucket

is $(1 + \epsilon)b$. If at least b balls land into a bucket, it becomes full, in which case the corresponding message bundle is recovered. If a bucket overflows, i.e. more than b balls land in it, then the balls in excess of the b balls needed are not **useful**. If the bucket is not full, then there is not enough information to recover the bundle. The question becomes calculating the probability that each bucket receives the required b balls.

A key parameter to this bundle based code is the bundle size b . If it is too big, then the encoding and decoding times are too big. If it is too small, then the probability of failure is too large. For example, consider the extreme example when the bundles are of size one. In this case, each code block would be the j^{th} message block with probability $1/m$. Receiving all m message blocks, becomes the classical coupon collector or occupancy problem. It is well known that $m \log m$ code blocks would need to be received, before one could expect to receive all m message blocks.

The probabilistic erasure code described in this paper will combine the bundle idea with the random matrix idea. Each code block is “about” one of the message bundles. Which bundle is chosen randomly. Here “about” means that it is a bitwise exclusive-or of a random subset of the message blocks in the bundle. Which subset of the message blocks is specified by a random vector $V_i \in \{0, 1\}^b$ and is often referred to as a **linear equation**. The following lemma bounds the success probability of this scheme.

Lemma 2 *If the bundles have size $b = \frac{2(1+12\epsilon)\ln(m/\delta)}{\epsilon^2}$, then the probability of not being able to recover the message is at most $\frac{3\delta}{b}$.*

The time per message bit of this scheme is then $\mathcal{O}(b) = \mathcal{O}\left(\frac{\ln(m/\delta)}{\epsilon^2}\right)$. When the message is large and ϵ and δ are reasonable constants, this is a marked improvement over $\mathcal{O}(m)$.

Proof of Lemma 2: Consider one of the m/b bundles. It is sufficient to prove that it can be recovered with probability $\frac{3\delta}{m}$. For purposes of simplification, ignore for a minute the problem of the equations being linearly dependent. The expected number of code blocks received about it is $(1 + \epsilon)b$. Chernoff-bounds (see for example [?]) give that the probability that fewer than b are received is at most $e^{-\frac{(\epsilon b)^2}{2(1+\epsilon)b}}$. Setting $b = \frac{2(1+\epsilon)\ln(m/\delta)}{\epsilon^2}$, gives $e^{-\frac{\epsilon^2}{2(1+\epsilon)} \frac{2(1+\epsilon)\ln(m/\delta)}{\epsilon^2}} = \frac{\delta}{m}$.

Lemma 1 proves that the probability of a random $(b + c \times b)$ -boolean matrix not having full rank is at at most $2 \cdot e^{-0.1c}$. Set c to $10 \ln(m/\delta)$ so that this probability is $\frac{2\delta}{m}$. Then let us redo the above calculations, but requiring that $b + 10 \ln(m/\delta)$ equations are received about the bundle. The probability that this many are not received remains $\frac{\delta}{m}$, by increasing the size of the bundle to $b = \frac{2(1+12\epsilon)\ln(m/\delta)}{\epsilon^2}$. ■

4 A Hierarchy of Bundle Sizes

The tradeoff, when setting the bundle size b , between the computation time and the probability of success, raises the question whether it is possible to get the best of both worlds by having a hierarchy of bundle sizes. The rest of the paper answers this question to the affirmative.

The scheme is as follows. Its parameters are $b_1 < b_2 < \dots < b_r$ and $0 < w_1, w_2, \dots, w_r \leq 1 + \epsilon$, where $\sum_i w_i = 1 + \epsilon$. The message M is first broken into bundles of size b_r . Then these bundles are broken into smaller sub-bundles of size b_{r-1} . These sub-bundles are broken further into yet smaller sub-sub-bundles of size b_{r-2} and so on. Let $B_{(i,j)} \subseteq [1..m]$ be the set of message blocks in

the j^{th} bundle of size b_i . Every code block is “about” one of the bundles $B_{\langle i,j \rangle}$, namely is linear combination of a randomly chosen subset of the message blocks in that bundle. Which bundle a code block is about is chosen randomly so that the expected number of code blocks received about the bundle $B_{\langle i,j \rangle}$ is $w_i b_i$. Note that the number of bundles of size b_i is m/b_i . Hence, the expected number of code blocks received about bundles of size b_i is $w_i m$. Recall that the total number of code blocks received is $(1 + \epsilon)m$. Hence, the constraint that $\sum_i w_i = 1 + \epsilon$.

Two possible ways of choosing which code blocks are about which bundles are as follows. The first way is simply to choose a bundle $B_{\langle i,j \rangle}$ independently at random for each code block by first choosing the size $i \in [1..r]$, where the size b_i is chosen with probability $w_i/(1 + \epsilon)$, and then choosing the bundle $j \in [1..m/b_i]$ each with probability b_i/m . The second way is simply to have $\frac{w_i}{(1+\epsilon)} \frac{b_i}{m} n$ of the n code blocks be about the bundle $B_{\langle i,j \rangle}$ and then to randomly permute the code blocks. This ensures that with every subset of $(1 + \epsilon)m$ code blocks, we expect get $w_i b_i$ code blocks about this bundle. The second scheme has three advantages. First it makes it easier for the code to be systematic, namely before permuting, the first m code blocks can be the message itself. If this is done, then the smallest bundle is of size $b_1 = 1$, with $w_1 = (1 + \epsilon)m/n$. The second advantage of the second distribution is that it makes it a little more likely that the number of code blocks received about a bundle is the expected number. The third advantage is that in practice, the code blocks might not in fact be randomly permuted, only interwoven in some fixed way. The hope then is that a random subset of the code blocks is received. On the other hand, the first scheme has the advantage that there is a simple local choice for each bundle. For analysis, this is the easier distribution to think about, but it does not really matter.

The encoding algorithm is simply boolean matrix multiplication $V \times M = E$ as described above and the decoding algorithm is using simple Gaussian elimination to solve the system $\hat{V} \times M = \hat{E}$ (with some extra care not to do work when there is known to be zeros in the matrix).

The rest of the paper consists of proving that for some setting of the parameters, the message can be recovered with probability $1 - \delta$ and the encoding and decoding times are as stated. We also prove the lower bound, that for no the setting of the parameters, are the encoding and decoding times for this scheme more than a constant factor better than we have achieved.

5 Bounding the Probability of Failure

This section bounds the probability of recovering the message in this bundle hierarchical probabilistic erasure-resilient code. This probability is best understood by defining the concepts of a code block being about, contributing to, and useful for a bundle $B_{\langle i,j \rangle}$. As defined above, a code block is **about** a bundle if the code block is a linear combination of a random subset of the blocks in the bundle. A code block **contributes** to a bundle if it is either about the bundle or it is useful to one of the bundles sub-bundles. A code block that contributes to a bundle is considered **useful** if it neither “overflows from the bundle” (the ball and bucket analogy is useful here) nor is linearly dependent on previous code blocks (this does not happen often). We will denote by $a_{\langle i,j \rangle}$, $c_{\langle i,j \rangle}$, and $u_{\langle i,j \rangle}$ the number of code blocks that are respectively about, contributing to, or useful for the bundle $B_{\langle i,j \rangle}$. More formally, suppose that we have calculated the number of useful code blocks for each of the bundles of size b_{i-1} . Consider a bundle $B_{\langle i,j \rangle}$ of size b_i . Dump into this “bucket” all the useful code blocks from smaller “buckets” for which the corresponding bundle is a sub-bundle, i.e. $B_{\langle i-1,j' \rangle} \subset B_{\langle i,j \rangle}$. These code blocks are the ones that are said to contribute to the bundle $B_{\langle i,j \rangle}$.

The number of them is $c_{\langle i,j \rangle} = a_{\langle i,j \rangle} + \sum_{(B_{\langle i-1,j' \rangle} \subset B_{\langle i,j \rangle})} u_{\langle i-1,j' \rangle}$. The number of code blocks that do not overflow from the bucket $B_{\langle i,j \rangle}$ is $\min(c_{\langle i,j \rangle}, b_i)$. Subtract off from these those code blocks that are linearly dependent on previous ones. This gives the number $u_{\langle i,j \rangle}$ that are **useful**. The bundle $B_{\langle i,j \rangle}$ is recovered if b_i useful code blocks are received about it.

We will want to predict the number of code blocks $\max(0, c_{\langle i,j \rangle} - b_i)$ that overflow from a bundle. To do this, we need to bound the expected number of code blocks $c_{\langle i,j \rangle}$ contributed to the bundle. This is done by assuming that all the code blocks about its sub-bundles are useful and hence are put into $B_{\langle i,j \rangle}$. By assuming this, we may be over counting. For different sizes of bundles, the same code block may be counted as having overflowed. However, the assumption makes the calculations much easier, namely $c_{\langle i,j \rangle} \leq \sum_{(B_{\langle i',j' \rangle} \subseteq B_{\langle i,j \rangle})} a_{\langle i',j' \rangle}$. The number of sub-bundles of $B_{\langle i,j \rangle}$ of size $b_{i'}$ is $\frac{b_i}{b_{i'}}$. Hence, the expected number of code blocks that contribute this bucket is $\exp(c_{\langle i,j \rangle}) \leq \sum_{i' \in [1..i]} \frac{b_i}{b_{i'}} \exp(a_{\langle i',j' \rangle}) \leq \sum_{i' \in [1..i]} \frac{b_i}{b_{i'}} w_{i'} b_{i'} = \left(\sum_{i' \in [1..i]} w_{i'} \right) b_i$.

Define q_i to be $1 - \sum_{i' \in [1..i]} w_{i'}$. Then the expected number of code blocks contributing to $B_{\langle i,j \rangle}$ is $(1 - q_i)b_i$. Recall that any number of code blocks over b_i overflow. The smaller q_i , the more you expect the bucket corresponding to $B_{\langle i,j \rangle}$ to be full and the more likely balls will overflow from the bucket. On the other hand, the larger b_i is, the less likely it is that the number of code blocks received will deviate far from the expected number. Hence, the larger b_i is, the smaller q_i is safely able to be.

We will now give the optimal setting of the parameters $b_1 < b_2 < \dots < b_r$ and w_1, w_2, \dots, w_r . Starting with the largest bundle, each bundle will be broken into two sub-bundles, giving $b_i = 2b_{i-1}$ for $i \in [3..r]$. If the code is to be systematic, then the message is part of the code and the smallest bundle is of size $b_1 = 1$, with $w_1 = (1 + \epsilon)m/n$. Any of these that are received are guaranteed to be useful, because by construction each message block appears only once in the code.

A very strong property that we are going to insist on is that with high probability for all bundles, except possibly those of the largest size, all code block contributed to it are useful to it. If this is not the case for even one code block, then we will assume the entire algorithm fails. The parameters for $i \in [2..r-1]$ are set to ensure this and Lemma 3 bounds this failure probability by $\delta \frac{r-1}{r}$. The key per Φ parameter setting is $q_i = \sqrt{\frac{2 \ln(m/\delta)}{b_i}}$. As explained above the larger b_i is, the smaller q_i needs to be to ensure the required property with high probability. The parameter w_i is then given by $(1 - q_i) - (1 - q_{i-1}) = q_{i-1} - q_i = \sqrt{\frac{2 \ln(m/\delta)}{b_{i/2}}} - \sqrt{\frac{2 \ln(m/\delta)}{b_i}}$ for $i \in [3..r-1]$. The second size b_2 is set so that $w_2 = q_1 - q_2 = (1 - (1 + \epsilon)m/n) - q_2$ has the same form $\sqrt{\frac{2 \ln(m/\delta)}{b_2/2}} - q_2$, giving $b_2 = \frac{4 \ln(m/\delta)}{(1 - (1 + \epsilon)m/n)^2}$.

What remains to fix is the size of the largest bundle. Assuming that all the code blocks contributed to smaller bundles are useful, there are only two remaining reasons that a code block is not useful. The first reason is that more than b_r code blocks contribute to a single large bundle. The second reason is that one of the code blocks that is about one of the largest bundles is linearly dependent with the earlier code blocks. This remaining scenario is identical to the scenario of when all the bundles are of the same size. Lemma 2 proved that if $b_r = 2(1 + 12\epsilon) \frac{\ln(m/\delta)}{\epsilon^2}$, then the probability of that the message cannot be recovered is at most $\frac{2\delta}{b_r}$, which is at most $\delta \frac{1}{r}$ (note that $q_i \leq 1$. Hence, $b_i \geq 2 \ln(m/\delta)$, which I am assuming is at least $= 3 \log \left(2(1 + 12\epsilon) \frac{\ln(m/\delta)}{\epsilon^2} \right) = 3 \log(b_r) \geq 3r$). Combining the two probabilities that the message is not recovered gives the

total of failure probability of δ . What remains is to prove the lemma.

Lemma 3 *The probability is at most $\delta \frac{r-1}{r}$ that there is a bundle, not of the largest size, that has a code block contributed to it that is not useful to it.*

Proof of Lemma 3: Consider a bundle $B_{\langle i,j \rangle}$, $i \in [2..r-1]$. The first step bounds the probability of the “balls overflowing the bucket”. The expected number of code blocks received about it or about one of its sub-bundles is $(1 - q_i)b_i$. Chernoff-bounds give that the probability that more than $(1 - q_i^2/2)b_i$ are received is at most $e^{-\frac{[(q_i - q_i^2/2)b_i]^2}{2[(1 - q_i)b_i]}} \leq e^{-\frac{q_i^2 b_i}{2}} = \frac{\delta}{m}$, when $q_i = \sqrt{\frac{2 \ln(m/\delta)}{b_i}}$. Now assume, that the bundle in fact gets fewer than $(1 - q_i^2/2)b_i$ code blocks about it.

The next step bounds the probability of one of the equations about this bundle is linearly dependent on the previous equations. Consider the i^{th} equation. If it is about a sub-bundle of $B_{\langle i,j \rangle}$ than we handle the case that it is dependent on the previous equations when considering that sub-bundle. If it is about the bundle $B_{\langle i,j \rangle}$ then it is a random vector of dimension b_i . The previous $i - 1$ equations span a sub-space of dimension $i - 1$. Hence, the probability that the i^{th} equation is within this sub-space is $2^{-(b_i - i + 1)}$. The probability that the $(1 - q_i^2/2)b_i$ equations are dependent is at most $\sum_{i \in [1..(1 - q_i^2/2)b_i]} 2^{-(b_i - i + 1)} \leq 2^{-(q_i^2/2)b_i} \leq \frac{\delta}{m}$.

It follows that the probability that one of the code blocks contributed to one of the $\frac{m}{b_i}$ bundles of size b_i is not useful is at most $\frac{2\delta}{b_i}$. The lemma considers the $r - 1$ sizes of bundles excluding the largest one. Hence, the probability is no more than $\frac{2\delta(r-1)}{b_i} \leq \frac{\delta(r-1)}{r}$ that there is a bundle, not of the largest size, that has a code block contributed to it that is not useful to it. ■

6 Bounding the Computation Time

This section bounds the expected computation time for encoding and decoding this bundle hierarchical probabilistic erasure-resilient code. Recall that encoding is done via boolean matrix multiplication $V \times M = E$, where V is the sparse matrix randomly chosen as described above and M is the matrix whose rows are the code blocks of the message. The code block E_i is obtained by adding (bitwise over GF[2]) the code blocks of the message specified by the row V_i . Adding two such blocks requires ℓ' exclusive-or operations, where $\ell' = \ell/32$ or $\ell/64$ is the number of computer words per block. The number of times such blocks need to be added is the number of non-zero entries in V . Below, the expected number of ones in this sparse matrix will be bounded to be $(6.8 + 53\epsilon) \frac{\ln(m/\delta)}{\epsilon} n$. Hence, the encoding time is $(6.8 + 53\epsilon) \frac{\ln(m/\delta)}{\epsilon} n \ell'$ or $(6.8 + 53\epsilon) \frac{\ln(m/\delta)}{\epsilon}$ operations per word of the encoding.

Decode is a little harder. Recall that it requires solving the system $\hat{V} \times M = \hat{E}$, where \hat{V} and \hat{E} are the rows of the matrices V and E corresponding to those code blocks that were received. Here again, the fact that \hat{V} is sparse should help. However, the problem is that the inverse of a sparse matrix is not necessarily sparse. Wiedemann [?] states that inverting an $(m \times m)$ -boolean matrix with only cm non-zero entries can be done in $\mathcal{O}(cm^2)$ bit operations. This is no improvement at all over Reed-Solomon codes. To improve the computation time, we take advantage of the fact that V has a bundle hierarchical structure.

Defⁿ: A $((1 + \epsilon)m \times m)$ -boolean matrix \hat{V} is said to have its rows bundled by $B_1, \dots, B_{(1+\epsilon)m} \subseteq [1..m]$, if each row is zero outside of its bundle B_i , i.e. $j \notin B_i$ implies that $\hat{V}_{\langle i,j \rangle} = 0$. These

bundles are said to have a hierarchical structure if the rows are partially ordered with respect to containment of the bundles, i.e. for $i < j$, either $B_i \subseteq B_j$ or $B_i \cap B_j = \emptyset$.

This bundle hierarchical structure improves the computation time for the following reason. During Gaussian elimination, the i^{th} row may be added to the j^{th} row in order to remove one non-zero entry from the j^{th} row. In a general sparse matrix, the non-zero entries of these rows do not necessarily fall in the same places. Hence, the j^{th} row will gain most of the non-zero entries of the i^{th} row. This means that a row operation generally doubles the number of non-zero entries in a row. The effect is that the number grows exponentially with the number of row operations. This provides intuition into why the inverse of a sparse matrix is not necessarily sparse. This exponential growth in the number of non-zero entries, however, does not occur when the matrix has the bundle hierarchical structure. By the definition of the partial order, B_i is either disjoint from or contained in B_j . In the first case, adding the i^{th} and the j^{th} row would not cancel any entries, hence these rows are never added together. In the second case, adding the i^{th} row to the j^{th} will change which entries in the bundle B_j are one, but will not contribute ones outside of the bundle. Hence, as the matrix \hat{V} is zeroed below the diagonal, the bundle hierarchical structure is maintained. On the other hand, when zeroing above the diagonal, this structure is destroyed, because the j^{th} row is added to the i^{th} row where $B_i \subset B_j$. In fact, the inverse of \hat{V} is not likely to be sparse. However, once the matrix is upper triangular, the system can be solved quickly.

To be more formal, the system $\hat{V} \times M = \hat{E}$ is solved using standard Gaussian elimination with some care taken so that unnecessary work is not done. The first step is to sort the rows of \hat{V} according to the partial order $B_i \subseteq B_j$. Associate with each row i , a pointer to the next row j for which $B_i \subseteq B_j$. A property of the partial order is that following this linked list of pointers starting at any row i will reach every row j for which $B_i \subseteq B_j$. (The same would not be true in reverse.)

The matrix \hat{V} is zeroed below the diagonal as follows. Suppose that for $i \in [1..m]$, all the entries of \hat{V} in the columns $[1..i - 1]$ are zero below the diagonal and are one on the diagonal. Delete any completely zero rows. Then move a one to the diagonal position $\langle i, i \rangle$ by switching the i^{th} column with a column containing a one in position $\langle i, j \rangle$. The remaining step is to add the i^{th} row to every row j below it that contains a one in the i^{th} column. This zeros the i^{th} column below the diagonal. If $B_i \not\subseteq B_j$, then there will not be a one in both locations $\langle i, i \rangle$ and $\langle j, i \rangle$. Hence, only the rows reached in the linked list starting at i are considered. Before doing this, a susinct list needs to be formed of the columns containing ones in the i^{th} row. This saves us from needing to scan the entire bundle B_i repeatedly for each row j considered. Because many of the entries of the i^{th} row have already been zeroed, this will be a considerable savings. Adding the i^{th} row to the j^{th} row then consists of flipping the bits of the j^{th} row of \hat{V} that are in the susinct list and of adding the i^{th} block of \hat{E} to the j^{th} block of \hat{E} .

The upper triangle is zeroed as follows. Suppose that the columns $[m..i + 1]$ are zero above the diagonal. It follows that the i^{th} row is zero except for the diagonal. This row is added to all the rows above it containing a one in the i^{th} column.

When calculating the computation time, let us separately consider those operations to the matrix \hat{V} and those to the matrix \hat{E} . The former does not depend on the length ℓ' of the blocks of \hat{E} , while the latter depends linearly on ℓ' . We recommend setting the parameter ℓ' to be large enough so that the latter time dominates the former.

Consider first the number of operations in \hat{E} . For each row operation in \hat{V} , two blocks of \hat{E} are added together. The number of row operations when zeroing below the diagonal is at most $\sum_{j \in [1..(1+\epsilon)m]} |B_j|$, because the i^{th} row is added to the j^{th} at most once and only if $i \in B_j$. The

number of row operations when zeroing above the diagonal is the number of one's that are above the diagonal of the upper triangular matrix. The bundle structure of the matrix does not change when zeroing below the diagonal. Hence, the number of ones is at most $\sum_{j \in [1..(1+\epsilon)m]} |B_j|$. Each entry $\langle i, j \rangle$ is either below or above the diagonal. Hence, the total number of row operations is at most $\sum_{j \in [1..(1+\epsilon)m]} |B_j|$. Note that this is the same as the decoding, except for the fact that there are $(1 + \epsilon)m$ instead of n rows.

The expectation of the sum $\sum_{j \in [1..(1+\epsilon)m]} |B_j|$ is bounded as follows. For $i \in [2..r-1]$, the expected number of rows that have bundles of size b_i is $w_i m = [q_{i-1} - q_i] m$. Therefore, excluding the largest sizes, the sum is

$$\begin{aligned}
& w_1 m \times 1 + \sum_{i \in [2..r-1]} [q_{i-1} - q_i] m \times b_i \\
&= \frac{(1 + \epsilon)m}{n} m + \sum_{i \in [2..r-1]} \left[\sqrt{\frac{2 \ln(m/\delta)}{b_{i-1}}} - \sqrt{\frac{2 \ln(m/\delta)}{b_i}} \right] b_i m \\
&\quad \text{but } b_{i-1} = \frac{b_i}{2} \text{ and } b_i = \frac{b_r}{2^{r-i}} \\
&\leq (1 + \epsilon)m + \sum_{i \in [2..r-1]} [\sqrt{2} - 1] \sqrt{2 \ln(m/\delta) b_i} m \leq \left[\sum_{i \in [1..\infty]} \left(\frac{1}{\sqrt{2}} \right)^i \right] [\sqrt{2} - 1] \sqrt{2 \ln(m/\delta) b_r} m \\
&= \left[\frac{1/\sqrt{2}}{1 - 1/\sqrt{2}} \right] [\sqrt{2} - 1] \sqrt{2 \ln(m/\delta) \left(2(1 + 12\epsilon) \frac{\ln(m/\delta)}{\epsilon^2} \right)} m \\
&\leq 2(1 + 6\epsilon) \frac{\ln(m/\delta)}{\epsilon} m
\end{aligned}$$

The expected number of rows that have the largest bundle size b_r is $w_r m = [q_{r-1} + \epsilon] m$. The sum is

$$\begin{aligned}
[q_{r-1} + \epsilon] m \times b_r &= \left[\sqrt{4 \ln(m/\delta) b_r} + \epsilon b_r \right] m \\
&= \left[2\sqrt{2}(1 + 6\epsilon) + 2(1 + 12\epsilon) \right] \frac{\ln(m/\delta)}{\epsilon} m
\end{aligned}$$

The total number of row operations is therefore $(6.8 + 53\epsilon) \frac{\ln(m/\delta)}{\epsilon} m$. The number of computer operations per row operation is ℓ' . Hence, the number of operations per word of the message is $(6.8 + 53\epsilon) \frac{\ln(m/\delta)}{\epsilon}$.

Now consider the number of operations within the matrix \widehat{V} that are needed to zero below the diagonal. Consider when the i^{th} row is added to those rows j for which $B_i \subseteq B_j$. The cost of adding the i^{th} row is the number of ones it contains (specified by the succinct list), which is b_i minus the number of entries that have already been zeroed. The number of entries that have been zeroed is the number of rows j before it for which $B_j \subseteq B_i$. The number of such rows which have bundle size b_k is $w_k b_k \frac{b_i}{b_k} = w_k b_i$. Hence, the total number is at least $\sum_{k \in [1..i-1]} w_k b_i = (1 - q_{i-1}) b_i$, concluding that the number of ones in the i^{th} row's succinct list is at most $q_{i-1} b_i$. The number of rows that the i^{th} row is added to is the number for which $B_i \subseteq B_j$ which is bounded by

$$\sum_{j \in [i..r]} w_j b_j = \sum_{j \in [i..r-1]} [q_{j-1} - q_j] b_j + [q_{r-1} + \epsilon] b_r \leq (6.8 + 53\epsilon) \frac{\ln(m/\delta)}{\epsilon}$$

Hence, the total cost of adding the i^{th} row to the necessary rows is at most $[q_{i-1} b_i] \left[(6.8 + 53\epsilon) \frac{\ln(m/\delta)}{\epsilon} \right]$. The sum of this cost over all the rows is bounded by

$$\begin{aligned}
& \sum_{i \in [1..r-1]} [(q_{i-1} - q_i)m] [q_{i-1} b_i] \left[(6.8 + 53\epsilon) \frac{\ln(m/\delta)}{\epsilon} \right] + [(q_{r-1} + \epsilon)m] [q_{r-1} b_r] \left[(4.8 + 41\epsilon) \frac{\ln(m/\delta)}{\epsilon} \right] \\
&= \sum_{i \in [1..r-1]} \left[(\sqrt{2} - 1) \sqrt{\frac{2 \ln(m/\delta)}{b_i}} m \right] \left[\sqrt{\frac{2 \ln(m/\delta)}{b_i/2}} b_i \right] \left[(6.8 + 53\epsilon) \frac{\ln(m/\delta)}{\epsilon} \right] \\
&\quad + \left(\sqrt{\frac{2 \ln(m/\delta)}{b_i/2}} \right)^2 b_r \left[(4.8 + 41\epsilon) \frac{\ln(m/\delta)}{\epsilon} \right] m \\
&\quad + \epsilon \sqrt{4 \ln(m/\delta) \cdot \left(2(1 + 12\epsilon) \frac{\ln(m/\delta)}{\epsilon^2} \right)} \left[(4.8 + 41\epsilon) \frac{\ln(m/\delta)}{\epsilon} \right] m \\
&= \left[\left[\sum_{i \in [1..r-1]} (8 + 62\epsilon) \right] + (8.6 + 81\epsilon) + (13.6 + 146\epsilon) \right] \frac{\ln^2(m/\delta)}{\epsilon} m \\
&\leq (8 + 62\epsilon) \log \left(\ln(m/\delta)/\epsilon^2 \right) \frac{\ln^2(m/\delta)}{\epsilon} m
\end{aligned}$$

$$\ell' \geq 1.2 \log \left(\ln(m/\delta)/\epsilon^2 \right) \ln(m/\delta).$$

To conclude this section, the encoding and decoding times are $(6.8 + 53\epsilon) \frac{\ln(m/\delta)}{\epsilon} \ell' n$ and $(8 + 62\epsilon) \log \left(\ln(m/\delta)/\epsilon^2 \right) \frac{\ln^2(m/\delta)}{\epsilon} m + (6.8 + 53\epsilon) \frac{\ln(m/\delta)}{\epsilon} \ell' m$ bit-wise exclusive-ors. In order to amortize the cost of inverting \hat{V} , we recommend setting the parameter ℓ' to be at least $1.2 \log \left(\ln(m/\delta)/\epsilon^2 \right) \ln(m/\delta)$ so that the second time dominates the first. Then the encoding time is $(6.8 + 53\epsilon) \frac{\ln(m/\delta)}{\epsilon}$ per word of the encoding and the decoding time is the same, but per word of the message.

7 The Lower Bound

In this section, we prove the lower bound, that for no the setting of the parameters, $b_1 < b_2 < \dots < b_r$ and w_1, w_2, \dots, w_r , are the encoding and decoding times for the bundle hierarchical probabilistic erasure-resilient code more than a constant factor better than we have achieved.

Before proving the lower bound, let us explain our initial intuition for what the parameters should have been. This will show the need for the lower bound, i.e. other setting might have given better times. This will also provide some intuition into what is needed for the lower bound. Our initial intuition was to have three types of bundles. The first type of bundle would be identical to the bundles b_1, \dots, b_{r-1} defined above. The idea with these is to have the expected number of code blocks contributed to them $(1 - q_i)b_i$ be small enough so that with high probability there is no over flow. The second type of bundle is not used in the above algorithm. It would have been of a medium size $b_* = \mathcal{O} \left(\frac{\ln(m/\delta)}{\epsilon} \right)$. The expected number of these would have been $w_* b_* = 1 b_*$ or even more $(1 + q_*)b_*$. Because their size is smaller than the large bundles b_r , we cannot with any reasonable probability predict how much underflow $\max(0, b_* - c_{\langle *, j \rangle})$ or overflow $\max(0, c_{\langle *, j \rangle} - b_*)$ there will be. Instead, consider a large number of these medium sized bundles, say the $\frac{b_r}{b_*}$ bundles that are included in the large bundle $B_{\langle r, j \rangle}$. By the law of large numbers, with high probability, the total underflow in these $u_{\langle r, j \rangle} = \sum_{j' \in [1..(b_r/b_*)]} \max(0, b_* - c_{\langle *, j' \rangle})$ will be close to its expected

value. The third type of bundle would be like our largest bundles $b_r = \mathcal{O}\left(\frac{\ln(m/\delta)}{\epsilon^2}\right)$. These are large enough so that with high probability at least $u_{\langle r,j \rangle}$ of them are received. These will fill in the gaps left by the underflow.

We do not use the bundles of the medium size for the following reason. The computation time for this three stage scheme was within a constant factor of that defined above. However, the time was dominated by the cost of processing the largest bundles. This cost could be brought down by a little at the expense of increasing the size of the medium bundles by a lot. In the end, the constant in front of the computation times was minimized by increasing the size of the medium bundle until it had the same size as the largest bundle.

The medium sized bundle presents an interesting problem in probabilities. Consider $\frac{b_r}{b_*}$ buckets each of size b_* . Throw balls at the buckets so that the expected number of balls landing in a single bucket is $(1 + q_*)b_*$. Let $c_{\langle *,j \rangle}$ be the number that lands in the j^{th} bucket and define $u = \sum_{j \in [1..(b_r/b_*)]} \max(0, b_* - c_{\langle *,j \rangle})$ to be the total underflow. What is the expected value of u and what is the probability that u is more than some amount z ?

[Possibly give sketch a sketch of this underflow problem]

Lemma 4 *Given any setting of the parameters $b_1 < b_2 < \dots < b_r$ and w_1, w_2, \dots, w_r , if the probability of not being able to recover the message from any $(1 + \epsilon)m$ of the code blocks is less than δ , then $\sum_{i \in [1..r]} w_i b_i$ is at least $\frac{\ln(m/\delta)}{7\epsilon}$.*

Recall that $\sum_{i \in [1..r]} w_i b_i$ is the number of operations per word the encoding to encode and the same per word the message to decode. Hence, the upper and the lower bound match within a constant factor.

Proof of Lemma 4: Consider a setting of the parameters for which $\sum_{i \in [1..r]} w_i b_i$ is less than $\frac{\ln(2m/\delta)}{3\epsilon}$. To begin we will assume that any set of no more than b_i code blocks about a bundle of size b_i will be linearly independent. In other words, for the lower bound, we consider only over and underflow.

In the encoding algorithm given, which code blocks are about which bundles is chosen by independently choosing for each code block a bundle so that the expected number of code blocks about bundles of size b_i is $w_i b_i$. For most of this proof, we will consider a different distribution. The new distribution chooses independently for each bundle the number of code blocks that will be about it. For bundles of size b_i , this number is chosen by the Poisson distribution with mean $w_i b_i$. After the lower bound has been proven for this distribution, we will prove that it is close enough to the original distribution that the same bound holds.

To begin we will select one of the bundle sizes b_i to take on the role of the medium bundle and one of the sizes to take on the role of the largest bundle. Denote these sizes respectively by $b_{\hat{i}}$ and by $b_{\hat{r}}$. We will assume that all the code blocks received about bundles that are smaller than $b_{\hat{i}}$ are useful. Hence, the expected number of code blocks contributed to a medium bundle is $\exp(c_{\langle \hat{i}, j' \rangle}) = \left(\sum_{i \in [1..\hat{i}]} w_i\right) b_{\hat{i}}$. Use $(1 + q_{\hat{i}})$ to denote this sum, where $q_{\hat{i}} \in [0..\epsilon]$. Generally, the bundle of the next largest size $b_{\hat{i}+1}$ will take on the role of the largest bundle. However, we want the flexibility to choose any value $q_{\hat{i}} \in [1..\epsilon]$. Hence, we may instead think of the bundle of size $b_{\hat{i}}$ as being two different bundles, where we expect to receive $w'_{\hat{i}} b_{\hat{i}}$ code blocks about the first and $(w_{\hat{i}} - w'_{\hat{i}}) b_{\hat{i}}$ code blocks about the second. In this case, let $(1 + q_{\hat{i}}) = \left(\sum_{i \in [1..(\hat{i}-1)]} w_i\right) + w'_{\hat{i}}$ and let both the “medium” and the “large” bundles be of size $b_{\hat{i}}$.

Consider one of the large bundles $B_{\langle \hat{r}, j \rangle}$ and its $\frac{b_{\hat{r}}}{b_i}$ sub-bundles of size b_i . (Note that $\frac{b_{\hat{r}}}{b_i}$ may actually be one.) Let $u_{\langle \hat{r}, j \rangle}$ denote the total underflow in these medium bundles, i.e. $u_{\langle \hat{r}, j \rangle} = \sum_{j' \in [1..(b_{\hat{r}}/b_i)]} \max(0, b_i - c_{\langle i, j' \rangle})$. Let $z_{\langle \hat{r}, j \rangle}$ denote the total number of code blocks received either about the large bundle $B_{\langle \hat{r}, j \rangle}$ or about one of its super bundles. The expected number of such code blocks is $\bar{z} = (w_i - w'_i)b_i + \sum_{i \in [(\hat{i}+1)..r]} w_i b_i$. The key observation is that if $u_{\langle \hat{r}, j \rangle} > z_{\langle \hat{r}, j \rangle}$, then there are not enough code words about the large bundle $B_{\langle \hat{r}, j \rangle}$ or about its super bundles to fill in the gaps left by the underflow within the $\frac{b_{\hat{r}}}{b_i}$ medium bundles. It would follow that the large bundle could not be recovered. There are many reasons that the full message is not fully recovered. However, the only one that we are going to consider is because $u_{\langle \hat{r}, j \rangle} > z_{\langle \hat{r}, j \rangle}$, for some $j \in [1..(m/b_{\hat{r}})]$.

An interesting thing about the expected number of code blocks about the super bundles of a medium bundle $\bar{z} = (w_i - w'_i)b_i + \sum_{i \in [(\hat{i}+1)..r]} w_i b_i$ is that it definitely less than $\sum_{i \in [1..r]} w_i b_i$, which by the assumption of the lemma is at most $\frac{\ln(2m/\delta)}{3\epsilon}$.

We are now going to choose $q_i \in [1..\epsilon]$ so that $q_i b_i \leq \bar{z} \leq q_i b_{\hat{r}}$. The proof that such a value for q_i exists is as follows. Think of b_i as a function of q_i , namely, b_i is that bundle size for which $\sum_{i \in [1..(\hat{i}-1)]} w_i < 1 + q_i \leq \sum_{i \in [1..\hat{i}]} w_i$. The function $q_i b_i$ is zero for $q_i = 0$ and then is monotone increasing, though not continuous. Similarly think of $\bar{z} = (w_i - w'_i)b_i + \sum_{i \in [(\hat{i}+1)..r]} w_i b_i$ as a function of q_i . This function is monotone decreasing, is continuous, and ends at zero for $q_i = \epsilon$. Therefore, the two functions $q_i b_i$ and \bar{z} either intersect at a point or cross at a discontinuity of the function $q_i b_i$. Either way, set q_i to the value at which they cross. In the first case, $q_i b_i = \bar{z}$ and the scenario is the one in which the same bundle is thought of both as the medium bundle and as the large bundle. Therefore, $q_i b_i = \bar{z} = q_i b_{\hat{r}}$. If the function \bar{z} crosses $q_i b_i$ at one of its discontinuities, then q_i is “at the boarder between two different bundles sizes”, i.e. $1 + q_i = \sum_{i \in [1..\hat{i}]} w_i$. If you increase q_i by an infinitesimal amount, then $q_i b_i$ jumps from $q_i b_i$ to $(q_i + o(1))b_{i+1}$ and bz falls between these values. The large bundle size $b_{\hat{r}}$ is considered to be b_{i+1} . In conclusion, $q_i b_i < \bar{z} < q_i b_{\hat{r}}$.

A property that will be useful later is that $q_i \geq \epsilon/2$. From above we have that $q_i b_{\hat{r}} \geq \bar{z}$. By definition, we have $\bar{z} = (w_i - w'_i)b_i + \sum_{i \in [(\hat{i}+1)..r]} w_i b_i$. Note that all of these bundle sizes are at least $b_{\hat{r}}$ and that $(w_i - w'_i) + \sum_{i \in [(\hat{i}+1)..r]} w_i = (1 + \epsilon) - (1 + q_i)$. Therefore, $\bar{z} \geq (\epsilon - q_i)b_{\hat{r}}$. It follows that $q_i \geq \epsilon/2$.

The next step is to bound the probability that there is a large bundle $B_{\langle \hat{r}, j \rangle}$ for which the total underflow $u_{\langle \hat{r}, j \rangle} = \sum_{j' \in [1..(b_{\hat{r}}/b_i)]} \max(0, b_i - c_{\langle i, j' \rangle})$ in its medium bundles is at least \bar{z} .

Consider first a single medium bundle. The expected number of code blocks contributed to it is $\exp(c_{\langle i, j' \rangle}) = (1 + q_i)b_i$. In order to have any underflow at all within this bundle, $c_{\langle i, j' \rangle}$ must be at least $q_i b_i$ less than its expectation. The probability of this is not all that different from the probability that $c_{\langle i, j' \rangle}$ is at least $2q_i b_i$ less than its expectation, in which case its underflow is at least $q_i b_i$. Hence, we will only consider events in which the medium bundles either have no underflow or have at least $q_i b_i$ underflow. Note that the probability of the latter is at least $e^{-\frac{(2q_i b_i)^2}{2(1+q_i)b_i}} \geq e^{-2q_i^2 b_i}$. In order to have a total of \bar{z} underflow, there must be $\frac{\bar{z}}{q_i b_i}$ bundles that have $q_i b_i$ underflow. Recall, we choose q_i so that $q_i b_i \leq \bar{z} \leq q_i b_{\hat{r}}$. Therefore, this number of bundles is at least one and no more than the number $\frac{b_{\hat{r}}}{b_i}$ of medium bundles per large bundle.

Partition the $\frac{m}{b_i}$ medium bundles of the message into groups of $\frac{\bar{z}}{q_i b_i}$ each. (Assume that $\frac{\bar{z}}{q_i b_i}$ is an integer and divides evenly into $\frac{b_{\hat{r}}}{b_i}$, so that each group is contained within one large bundle.)

The probability that a medium bundle has $q_i b_i$ underflow is at least $e^{-2q_i^2 b_i}$. Because of the new distribution, $c_{\langle i, j' \rangle}$ is independent across the different medium bundles. Therefore, the probability that each of the medium bundles within a single group has at least this underflow is at least $\left[e^{-2q_i^2 b_i} \right]^{\frac{\bar{z}}{q_i b_i}} = e^{-2q_i \bar{z}}$. The probability that this occurs within one of the $\frac{m}{b_i} \frac{q_i b_i}{\bar{z}} = \frac{q_i m}{\bar{z}}$ groups is at least $1 - \left[1 - e^{-2q_i \bar{z}} \right]^{\frac{q_i m}{\bar{z}}} \geq \frac{q_i m}{\bar{z}} e^{-2q_i \bar{z}} - \left[\frac{q_i m}{\bar{z}} e^{-2q_i \bar{z}} \right]^2$. Recall that $q_i \in [(\epsilon/2), \epsilon]$ and $\bar{z} \leq \frac{\ln(2m/\delta)}{3\epsilon}$ $\leq \frac{\ln(\epsilon m / (3\bar{z}\delta))}{2\epsilon}$. This give the probability to be at most $3\delta - [3\delta]^2 \geq \epsilon\delta$. Now note that if there is a group of $\frac{\bar{z}}{q_i b_i}$ medium bundles each of which has $q_i b_i$ underflow, then there is a large bundle $B_{\langle \hat{r}, j \rangle}$ for which the total underflow $u_{\langle \hat{r}, j \rangle}$ in its medium bundles is at least \bar{z} . Hence, the probability that this happens is at least $\epsilon\delta$.

What remains is to bound the probability that the message cannot be recovered. The probability is at least $\epsilon\delta$ that there is a large bundle $B_{\langle \hat{r}, j \rangle}$ for which the total underflow $u_{\langle \hat{r}, j \rangle}$ in its medium bundles is at least \bar{z} . Independent of this, the probability is at least $\frac{1}{e}$ that the number of code blocks $z_{\langle \hat{r}, j \rangle}$ received about this large bundle $B_{\langle \hat{r}, j \rangle}$ or about one of its super bundles is less than its expectation \bar{z} . Hence, the probability that both of these things occur is at least δ . As stated above, if this does occur then there are not enough code words about the large bundle $B_{\langle \hat{r}, j \rangle}$ or about its super bundles to fill in the gaps left by the underflow within the $\frac{b_{\hat{r}}}{b_i}$ medium bundles. It follows that the large bundle cannot be recovered.

This completes the lower bound when the new distribution is used. What remains to prove is that the new distribution is close enough to the original one that the same bound holds. To do this, consider a setting of the parameters for which $\sum_{i \in [1..r]} w_i b_i$ is less than $\frac{\ln(m/\delta)}{7\epsilon}$, as is needed to prove the lemma. We will apply the lower bound just proved, on slightly altered parameters, namely $\delta' = 2\delta$, $\epsilon' = 2\epsilon$ and for $i \in [1..r]$, $w'_i = \frac{(1+\epsilon')}{(1+\epsilon)} w_i$. Note that for these parameters $\sum_{i \in [1..r]} w'_i = (1+\epsilon')$ as it should be and $\sum_{i \in [1..r]} w'_i b_i \leq \frac{(1+\epsilon')}{(1+\epsilon)} \frac{\ln(m/\delta)}{7\epsilon} \leq \frac{\ln(2m/\delta')}{3\epsilon'}$ as is needed to apply the above lower bound.

Let *old* denote the original distribution with the original parameters, i.e. independently for each code block, a bundle is chosen so that the expected number of code blocks about bundles of size b_i is $w_i b_i$. Let *new* denote be the new distribution with the new parameters, i.e. for each bundle of size b_i , the number of code words about it is chosen according to the Poisson distribution with mean $w'_i b_i$. Let E denote the event that message cannot be fully recovered. The lower bound above proves that $\Pr_{new}[E] \geq \delta'$. Our goal now is to prove that $\Pr_{old}[E] \geq \delta$.

The key observation is that the only difference between the distributions is that in the original distribution, the total number code blocks received is definitely $(1+\epsilon)m$, while in the new one the number received is a random variable. Let K denote this number received. It happens that if one takes the new distribution with the added constraint that $K = (1+\epsilon)m$, then one gets the original distribution, i.e. $\Pr_{old}[E] = \Pr_{new}[E \mid K = (1+\epsilon)m]$. Increasing the number of code blocks received only increases the probability of recovering the message. Therefore, $\Pr_{old}[E] \geq \Pr_{new}[E \mid K \geq (1+\epsilon)m]$. The variable K has the Poisson distribution with mean $(1+2\epsilon)m$. Therefore, $\Pr_{new}[K < (1+\epsilon)m] \leq e^{-\frac{(\epsilon m)^2}{(1+2\epsilon)m}}$. We can now complete the proof.

$$\begin{aligned} 2\delta &= \delta' \leq \Pr_{new}[E] \\ &= \Pr_{new}[E \mid K \geq (1+\epsilon)m] \cdot \Pr_{new}[K \geq (1+\epsilon)m] \\ &\quad + \Pr_{new}[E \mid K < (1+\epsilon)m] \cdot \Pr_{new}[K < (1+\epsilon)m] \end{aligned}$$

$$\leq \Pr_{old}[E] \cdot 1 + 1 \cdot e^{-\frac{(\epsilon m)^2}{(1+2\epsilon)m}}.$$

Finally, we are assuming that $e^{-\frac{(\epsilon m)^2}{(1+2\epsilon)m}} \leq \delta$. ■