ON APPLICATIONS OF PUNCTURING IN ERROR-CORRECTION CODING

A Thesis
Presented to
The Academic Faculty

by

Demijan Klinc

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
May 2011

ON APPLICATIONS OF PUNCTURING IN ERROR-CORRECTION CODING

ACKNOWLEDGEMENTS

The path that culminated in this work has been long, fulfilling and many times frustrating. I have had a good fortune of having been in company of many good people and friends whom I will be forever indebted for making my stay in Atlanta a pleasurable one.

I would like to thank my advisor Steven McLaughlin. He decided to give me an opportunity in his group and he has always been there for me, offering unconditional support. It has been a pleasure to be around him all these years.

My deepest gratitude goes to Jeongseok Ha, with whom I have continued to collaborate intensely during my graduate work. His friendship, positive attitude, and deep knowledge of coding theory and computer science have inspired me, fed me with ideas and kept me going through hard times. The 12 hour time difference between us (thank you, Skype) has caused some difficulties, but it is safe to say that I cannot imagine finishing this thesis without his help.

The summer I spent at IBM Research Labs was one of the most fulfilling research experiences in my life. I had an opportunity to work on various challenging problems in a multidisciplinary team with extremely smart and driven people. Some of the work I performed there also forms part of this thesis and I would like to thank my mentor, Ashish Jagmohan, and other collaborators: Carmit Hazay, Tal Rabin, and Hugo Krawczyk for letting me be part of the team.

And my family. They have always been with me, supported me, motivated and loved me. I could not have come to this point without them. A special acknowledgement goes to my Dad, who took the time to read the entire thesis in great detail and provided numerous corrections and comments to improve the overall quality.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . iii

LIST OF TABLES . . . vi

LIST OF FIGURES . . . vii

SUMMARY . . . x

I LOW-DENSITY PARITY-CHECK CODES . . . 1

- 1.1 Error Control Coding . . . 1
- 1.2 Linear Block Codes . . . 2
- 1.3 LDPC Code Fundamentals . . . 3
  - 1.3.1 The Tanner Graph . . . 3
  - 1.3.2 Regular LDPC Codes . . . 5
  - 1.3.3 Irregular LDPC Codes . . . 6
- 1.4 Constructing of LDPC Codes . . . 7
- 1.5 Encoding LDPC Codes . . . 8
- 1.6 Decoding LDPC Codes . . . 8
  - 1.6.1 Probability Domain Decoding . . . 8
  - 1.6.2 Log-likelihood Domain Decoding . . . 13

II PUNCTURED BINARY LDPC CODES . . . 16

- 2.1 Basic Ideas . . . 17
- 2.2 Criteria for Selecting Punctured Nodes . . . 21
- 2.3 The Grouping Algorithm . . . 25
- 2.4 Simulation Results . . . 29
- 2.5 Layered BP Decoding for Punctured LDPC Codes . . . 33
- 2.6 Concluding Remarks . . . 38

III PUNCTURED AND SHORTENED NON-BINARY LDPC CODES . . . 40

- 3.1 Non-binary LDPC Codes . . . 40
- 3.2 The Puncturing Problem . . . 43
- 3.3 Puncturing and Shortening Over the BEC . . . 44
  - 3.3.1 Density Evolution Over the BEC . . . 46

3.4 Optimization of Puncturing and Shortening Distributions 48
3.4.1 Simulation Results 49
3.4.2 Highest Achievable Rate 54
3.5 Puncturing and Shortening at Finite (Short) Block Lengths 56
3.5.1 Puncturing Algorithm 59
3.5.2 Shortening Algorithm 60
3.5.3 Simulation Results 60
3.6 Concluding Remarks 69
IV PHYSICAL LAYER SECURITY 71
4.1 Channel Model and Basic Notions 71
4.2 Coding for Security 76
4.2.1 Asymptotic Analysis 76
4.2.2 Optimized Puncturing Distributions 81
4.2.3 Reducing SNR Loss 81
4.2.4 Finite Block Lengths 85
4.3 Efficient Encoding 85
4.4 System Aspects 88
4.5 Concluding Remarks 90
V COMPRESSION OF DATA ENCRYPTED WITH BLOCK CIPHERS 91
5.1 Preliminaries 92
5.2 Compressing Block-cipher Encryption 96
5.2.1 Cipher Block Chaining 97
5.2.2 Output Feedback 100
5.2.3 Cipher Feedback 101
5.2.4 Electronic Code Book 103
5.3 Compression Performance 106
5.4 Concluding Remarks 108
APPENDIX A SECURITY OF THE CBC COMPRESSION SCHEME AND COMPRESSIBILITY OF BLOCK CIPHERS IN ECB MODE 110
REFERENCES 121

LIST OF TABLES

1 Group distribution of the variable nodes, block length $N = 1024$. 30

2 Group distribution of the variable nodes, block length $N = 4096$. 31

3 The distribution of check nodes in layers. The highest level of recoverability among the punctured variable nodes is 5. 37

4 Optimized puncturing distributions obtained via differential evolution. Mother code is a regular (2,4), rate 0.5, LDPC code over GF(2^6). 50

5 Optimized shortening distributions obtained via differential evolution. Mother code is a regular (2,4), rate 0.5, LDPC code over GF(2^6). 51

6 Number of zero components in the initial LLRV for a punctured symbol in GF(2^6) 58

7 The puncturing pattern achieving code rate 0.91 by levels of recoverability of variable nodes for the considered binary and GF(2^6) LDPC code. 62

8 The breakdown of puncturing patterns obtained by the proposed algorithm for the non-binary LDPC code over GF(2^6). Except in the bottom row, the listed numbers indicate the number of variable nodes. 63

9 Asymptotically optimized puncturing patterns for the non-binary LDPC code over GF(2^6). 64

10 The proposed shortening patterns. 67

11 Puncturing distributions optimized for security. $P^E_{e,\min}$ was set to 0.49. . . . 81

12 Optimized puncturing distributions for $R_s = R_d$ codes and $P^E_{e,\min} = 0.49$. . 84

13 Asymptotic and empirical attainable gaps. 88

14 Attainable compression rates for block length $m = 128$ bits. 108

15 Attainable compression rates for $m = 1024$ bits. 108

LIST OF FIGURES

1 A parity-check matrix and its Tanner graph. 4

2 A length 4 cycle. 5

3 Check node message update. 11

4 Variable node message update. 12

5 A 1-SR variable node. Void and filled circles are punctured and unpunctured variable nodes, respectively. 18

6 A 3-SR variable node. Void and filled circles denote punctured and unpunctured variable nodes, respectively. 18

7 A recovery tree of a 3-SR node. Void and filled circles denote punctured and unpunctured variable nodes, respectively. 20

8 A stopping set. The four left most variable nodes form a stopping set. . . . 20

9 The plot of the function $\phi(x)$. 25

10 Comparison between the intentionally and randomly punctured LDPC code; the mother code has a block length of 1024. The curves from left to right correspond to: the unpunctured mother code and the punctured codes at rates 0.6, 0.7, and 0.8. 31

11 Comparison between the intentionally and randomly punctured LDPC code; the mother code has a block length of 4096. The curves from left to right correspond to: the unpunctured mother code and the punctured codes at rates 0.6, 0.7, and 0.8. 32

12 The distribution of required number of iterations for intentional and random puncturing over the AWGN channel at $R = 0.8$ and BER $= 10^{-5}$. 33

13 A Tanner graph split into two check nodes layers: L_1 and L_2. 34

14 Layers L_1 and L_2 of the Tanner graph in Figure 13 and their subgraphs. . . 35

15 Recovery tree of a 3-SR node where $p_2, p_3 \in L_1$, $p_4 \in L_2$, $p_5 \in L_3$, $p_1 \in L_{K+1}$, and $K \geq 3$. 36

16 BERs of punctured LDPC codes at code rates 0.5, 0.6, 0.7, 0.8 and 0.9 over an AWGN channel with the maximum number of iterations set to 15. The filled squares, unfilled circles, and unfilled squares represent the BERs of conventional BP decoding, layered BP decoding with random layering, and proposed layering, respectively. 38

17 The required E_b/N_0 values for a BER of 10^{-4} at the code rate 0.9 versus the maximum number of iterations, where the LDPC codes are the same as for Figure 16. 39

18 Non-binary variable node with its binary representation. 41

19 An example of a non-binary Tanner graph. 42

20 Symbol-wise and bit-wise puncturing. 43

21 Performance comparison between the proposed optimized codes (DiffEv), symbol-wise shortening and puncturing (6bps), and uniform bit-wise shortening and puncturing (3bps). 52

22 Frame-error-rate (FER) performance of punctured LDPC codes; rates from right to left: 0.5 (unpunctured), 0.52, 0.56, 0.58, and 0.62. The dashed lines represent the corresponding thresholds. 53

23 FER performance of punctured LDPC codes; rates from right to left: 0.66, 0.71, 0.76, 0.83, and 0.91. The dashed lines represent the corresponding thresholds. 54

24 FER performance of shortened LDPC codes; rates from right to left: 0.09, 0.17, 0.23, and 0.29. The dashed lines represent the corresponding thresholds. 55

25 FER performance of shortened LDPC codes; rates from right to left: 0.33, 0.38, 0.41, 0.44, and 0.47. The dashed lines represent the corresponding thresholds. 55

26 Performance of regular rate-adaptive non-binary LDPC codes over GF(2^4), GF(2^6) and GF(2^{10}). 56

27 Cut-off rate vs. parameter m. 57

28 Performance comparison between four differently chosen puncturing patterns at rate 0.83. 62

29 Performance comparison between symbol-wise, asymptotically optimized, and proposed puncturing at rates 0.62, 0.71 and 0.83 (from right to left) over the BEC channel. 65

30 Performance comparison between symbol-wise, asymptotically optimized, and proposed puncturing at rates 0.62, 0.71 and 0.83 (left to right) over the AWGN channel. 65

31 Performance of punctured non-binary and binary codes at rates 0.62 – 0.91 over the BEC channel. 66

32 Performance of punctured non-binary and binary codes at rates 0.62 through 0.91 over the AWGN channel. 66

33 Performance comparison between symbol-wise, asymptotically optimized, and proposed shortening at rates 0.23, 0.33 and 0.44 (from right to left) over the BEC channel. 68

34 Performance comparison between symbol-wise, asymptotically optimized, and proposed shortening at rates 0.23, 0.33 and 0.44 (left to right) over the AWGN channel. 68

35 Performance of shortened non-binary and binary codes at rates 0.23 through 0.44 over the BEC channel. 69

36 Performance of shortened non-binary and binary codes at rates 0.23 through 0.44 over the AWGN channel. 70

37 The classic Shannon model for secure communication. 71

38 The Gaussian wiretap channel. 73

39 The security gap. A typical BER vs. SNR performance curve of an error-correction code is shown. $\mathrm{SNR}_{B,\min}$ is the threshold for reliability (between Alice and Bob) and $\mathrm{SNR}_{E,\max}$ is the point below which Eve has a very high error rate, typically close to 0.5. The security gap (in dB) is the SNR difference between $\mathrm{SNR}_{B,\min} - \mathrm{SNR}_{E,\max}$ that must be maintained between Bob and Eve in order to achieve both reliability and security constraints in conditions a) and b). 75

40 Block diagram of the proposed encoder. 76

41 Density of a check-to-variable node message obtained by GA and DE when the decoder operates below the reliability threshold. 79

42 Eve's BER performance when operating below the reliability threshold $\mathrm{SNR}_{B,\min}$ and message bits are (i) transmitted and (ii) punctured. 80

43 The performance comparison between random and optimized puncturing. The probability $P^E_{e,\min} = 0.49$. 82

44 SNR loss comparison between codes with $R_s = R_d$ and $R_s < R_d$. 84

45 Security gap comparison between $R_s = R_d$ and $R_s < R_d$ codes. 85

46 BER vs. security gap at block length 2364. 86

47 Lower-triangular parity-check metrix suitable for efficient encoding. 86

48 Lossless source coding with decoder side-information. 93

49 Traditional system with compression preceding encryption. 93

50 System with encryption subsequent to compression. 94

51 Cipher block chaining. 98

52 Compressor. 99

53 Joint decryption and decoding at the receiver. It is performed serially from right to left. 100

54 Output feedback. 101

55 Cipher feedback. 102

56 Joint decryption and decoding in CFB mode at the receiver is performed serially from left to right. 103

57 Electronic Codebook. 104

SUMMARY

This thesis investigates applications of puncturing in error-correction coding and physical layer security with an emphasis on binary and non-binary LDPC codes.

Theoretical framework for the analysis of punctured binary LDPC codes at short block lengths is developed and a novel decoding scheme is designed that achieves considerably faster convergence than conventional approaches. Subsequently, optimized puncturing and shortening is studied for non-binary LDPC codes over binary input channels. Framework for the analysis of punctured/shortened non-binary LDPC codes over the BEC channel is developed, which enables the optimization of puncturing and shortening patterns. Insight from this analysis is used to develop algorithms for puncturing and shortening of non-binary LDPC codes at finite block lengths that perform well. It is confirmed that symbol-wise puncturing is generally bad and that bit-wise punctured non-binary LDPC codes can significantly outperform their binary counterparts, thus making them an attractive solution for future communication systems; both for error-correction and distributed compression.

Puncturing is also considered in the context of physical layer security. It is shown that puncturing can be used effectively for coding over the wiretap channel to hide the message bits from eavesdroppers. Further, it is shown how puncturing patterns can be optimized for enhanced secrecy. Asymptotic analysis confirms that eavesdroppers are forced to operate at BERs very close to 0.5, even if their signal is only slightly worse than that of the legitimate receivers. The proposed coding scheme is naturally applicable at finite block lengths and allows for efficient, almost-linear time encoding.

Finally, it is shown how error-correcting codes can be used to solve an open problem of compressing data encrypted with block ciphers such as AES. Coding schemes for multiple chaining modes are proposed and it is verified that considerable compression gains are attainable for binary sources.

CHAPTER I

LOW-DENSITY PARITY-CHECK CODES

1.1 Error Control Coding

Digital signals composed of binary digits, 0s and 1s, are used to represent and communicate different types of information, like text, video, audio, pictures, etc. Unfortunately, during storage, transmission, and processing of binary data, errors may be unintentionally introduced — a 1 may be changed to a 0 or vice versa.

In case of data transmission, each transmitted bit is received in the presence of noise or distortion and only an indication of the bit's value is obtained. While the number of errors may be relatively low, even a small number of errors can result in the data being unusable. In order to provide a mechanism to check for errors and to correct them, binary data can be coded to introduce carefully designed redundancy. Coding of a unit of data produces what is commonly referred to as a *codeword*. Due to its built-in redundancy, a codeword often includes more bits than the input unit of data from which the codeword was produced.

When signals arising from transmitted codewords are received and processed, the redundant information included in the codeword can be used to identify and correct errors from the received signal in order to recover the original data unit. Such error correcting can be implemented as part of a decoding process. In the case of unrecoverable errors, the decoding process may produce some indication that the original data cannot be fully recovered, which can be used to initiate retransmission of the data.

While redundancy can increase the reliability of data to be stored or transmitted, it comes at the cost of storage space and the use of valuable communications bandwidth. Accordingly, it is desirable to add redundancy in an efficient manner to maximize the amount of error correction capacity gained for a given amount of redundancy introduced in the data.

1.2 Linear Block Codes

A *block code* is a proceedure for converting a sequence $\boldsymbol{m} = [m_1, \ldots, m_K]$ of K source bits, called the *information block*, into a sequence $\boldsymbol{c} = [c_1, \ldots, c_N]$ of N bits, called the *codeword*, where $N > K$. The bits in the information block are also referred to as *information bits*. The ratio

$$R = \frac{K}{N} \tag{1}$$

is called *code rate*[1].

A block code is *linear* if there exists a generator matrix $\boldsymbol{G}$ with K rows and N columns, where $G_{ij} \in \{0, 1\}$, such that for any $\boldsymbol{m}$

$$\boldsymbol{c} = \boldsymbol{m}\boldsymbol{G}. \tag{2}$$

Block codes where the information bits always form a subset of codeword bits, i.e. $\boldsymbol{c} = [\boldsymbol{m}, \boldsymbol{p}]$, are referred to as *systematic*. The codeword bits, which are not information bits are called *parity bits*.

The set of all codewords contains exactly 2^K elements, since there exist 2^K information blocks and the mapping is 1-to-1. As $N > K$, not every sequence of N bits will be a codeword. If during transmission some codeword bits are flipped, the resulting $\boldsymbol{c}$ may not be a codeword anymore. As a tool for codeword recognition we introduce a *parity-check matrix* $\boldsymbol{H}$ with M rows and N columns, where $H_{ij} \in \{0, 1\}$, such that

$$\boldsymbol{H}\boldsymbol{c}^T = \boldsymbol{0} \tag{3}$$

is fulfilled if and only if $\boldsymbol{c}$ is a codeword[2]. The product of each row in $\boldsymbol{H}$ with $\boldsymbol{c}^T$ checks whether $\boldsymbol{c}$ fulfills one of M parity-check constraints. If the product is 0, the parity-check constraint is fulfilled, otherwise it is not. Thus, $\boldsymbol{c}$ is a codeword if and only if *all* parity-check constraints are fulfilled.

From Eqs. (2) and (3) we get the following relation between the generator matrix and the parity-check matrix:

$$\boldsymbol{H}\boldsymbol{G}^T = \boldsymbol{0}. \tag{4}$$

[1]In this thesis, we sometimes refer to it as rate.

[2]$\boldsymbol{c}^T$ denotes $\boldsymbol{c}$ transposed.

Example 1.1. *A systematic linear block code with $K = 4$ and $N = 7$ is defined by the following generator matrix:*

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

A 1 in the i-th row and j-th column indicates that the i-th bit in the information block influences the j-th codeword bit. The first four columns build an identity matrix, making the code systematic. Using Eq. (4) we calculate the parity-check matrix:

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

Each row represents a parity-check constraint that every codeword has to fulfill according to Eq. (3). A 1 in the i-th row and the j-th column indicates that the j-th bit in the codeword is involved in the i-th parity-check constraint.

A linear block code with a sparse parity-check matrix, that is a matrix with a small number of non-zero elements, it is called a *low-density parity-check* (LDPC) code.

1.3 LDPC Code Fundamentals

LDPC codes were proposed in 1962 by Gallager [17], along with an elegant iterative decoding scheme whose complexity grows only linearly with block length. They were largely forgotten for several decades until their huge potential was discovered in the 1990s, following the discovery of turbo codes. Most contributions in this thesis are tightly connected with LDPC codes; therefore, it is appropriate to focus some attention on their fundamentals.

1.3.1 The Tanner Graph

Every LDPC code is uniquely specified by its parity-check matrix $\boldsymbol{H}$ or equivalently, by means of the *Tanner graph* [68], as illustrated in Figure 1. The Tanner graph consists of two types of nodes: *variable nodes* and *check nodes*, and edges between them. Consider an LDPC code defined by

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

and its corresponding Tanner graph. Each variable node, depicted by a circle, represents

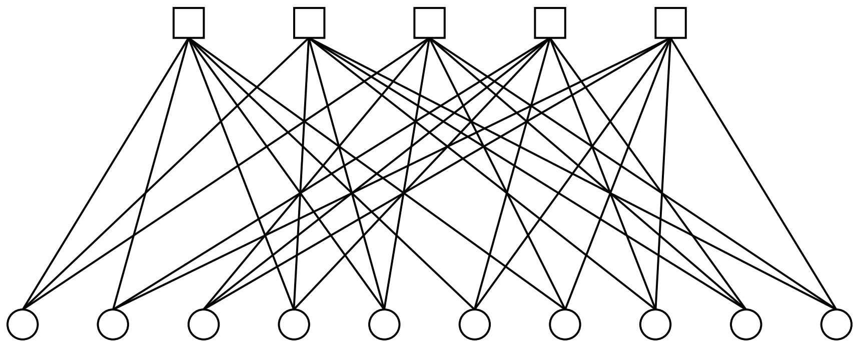

Figure 1: A parity-check matrix and its Tanner graph.

one bit of a codeword and every check node, depicted by a square, represents one parity-check constraint. Hence, the Tanner graph contains N variable nodes and M check nodes. The i-th variable node is connected to the j-th check node if and only if $H_{ij} = 1$. If d edges emanate from a node, variable or check node, we say that node has *degree* d. Since there are no direct connection between any two nodes of the same type, the Tanner graph is said to be *bipartite*. Tanner graphs can also serve as a nice visualization tool for a variety of issues concerning LDPC codes.

A *cycle* is a path in the Tanner graph that begins and ends at the same node, whereby every edge is traversed only once; the *length of a cycle* is the number of its edges. Usually, the Tanner graphs of LDPC codes contain many cycles of different lengths. A cycle of smallest length in the Tanner graph is called *girth*. Since Tanner graphs are bipartite, the smallest cycle has length 4, like shown in Figure 2. It is desired that LDPC codes do not have any cycles of length 4, as such codes yield poor performance.

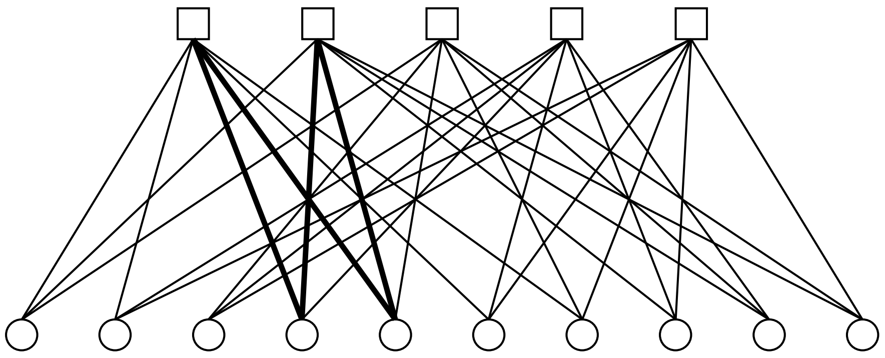

Figure 2: A length 4 cycle.

1.3.2 Regular LDPC Codes

An LDPC code whose parity-check matrix contains the same number d_v of non-zero elements in each column and the same number d_c of non-zero elements in each row, is said to be *regular* or more precisely a (d_v, d_c) regular LDPC code. The first LDPC codes introduced by Gallager were regular.

In a (d_v, d_c) regular LDPC code each codeword bit is subject to d_v parity-check constraints and each parity-check constraint involves d_c codeword bits. We say that the pair $(d_v,\ d_c)$ designates an *ensemble* of regular LDPC codes, since generally one can construct numerous different parity-check with d_v/d_c non-zero elements in each column/row. Each such parity-check matrix is called an *instance* of the ensemble. It is shown in [17] that best performing regular LDPC codes have $d_v = 3$.

The number of all non-zero elements in a parity-check matrix $\boldsymbol{H}$ of a regular LDPC code equals Md_c (or equivalently, Nd_v); hence, $M/N = d_v/d_c$ and the rate of a regular LDPC code can be expressed as

$$R = \frac{K}{N} = \frac{N-M}{N} = 1 - \frac{M}{N} = 1 - \frac{d_v}{d_c}. \tag{5}$$

1.3.3 Irregular LDPC Codes

An LDPC code that is not regular is designated as *irregular*. An irregular LDPC code can have a parity-check matrix $\boldsymbol{H}$ with different numbers of non-zero elements in each column/row, as long as it remains sparse. An ensemble of irregular LDPC codes is defined by the *degree distribution pair*

$$\begin{aligned} \lambda(x) &= \sum_{i=2}^{d_v} \lambda_i x^{i-1} && (6) \\ \rho(x) &= \sum_{i=2}^{d_c} \rho_i x^{i-1}, && (7) \end{aligned}$$

where λ_i is the fraction of edges emanating from variable nodes of degree i, ρ_i is the fraction of edges emanating from check nodes of degree i, and d_v and d_c are the maximum variable node and check node degrees, respectively. Since λ_i and ρ_i designate fractions of edges, we say that the above degree distribution pair is from the edge perspective.

If there are N variable nodes, the number of variable nodes of degree i is

$$\begin{aligned} N_i &= N \frac{\lambda_i / i}{\sum_{j=2}^{d_v} \lambda_j / j} \\ &= N \frac{\lambda_i / i}{\int_0^1 \lambda(x) dx}, \end{aligned}$$

and the total number of edges is

$$\begin{aligned} E &= N \sum_{i=2}^{d_v} \frac{\lambda_i}{\int_0^1 \lambda(x) dx} \\ &= N \frac{1}{\int_0^1 \lambda(x) dx}. && (8) \end{aligned}$$

Alternatively, if there are M check nodes, we get

$$E = M \frac{1}{\int_0^1 \rho(x) dx}. \qquad (9)$$

By combining Eqs. (8) and (9) we get the formula for the rate of an irregular LDPC code

$$R = 1 - \frac{M}{N} = 1 - \frac{\int_0^1 \lambda(x) dx}{\int_0^1 \rho(x) dx}. \qquad (10)$$

Sometimes it is convenient to have variable and check node distributions from the node perspective

$$\Lambda(x) = \sum_{i=2}^{d_v} \Lambda_i x^{i-1} \tag{11}$$

$$P(x) = \sum_{i=2}^{d_c} P_i x^{i-1}, \tag{12}$$

where Λ_i and P_i are the fraction of variable and check nodes of degree i, respectively. Using Eqs. (6) and (7) we get

$$\Lambda_i = \frac{\lambda_i/i}{\sum_{j=2}^{d_v} \lambda_j/j} \tag{13}$$

$$P_i = \frac{\rho_i/i}{\sum_{j=2}^{d_c} \rho_j/j}. \tag{14}$$

The design of irregular LDPC codes is more flexible due to the absence of constraints and they have proved to perform much better then the regular LDPC codes [60, 61]. For simplicity, we restrict subsequent analysis to regular LDPC codes, but the ideas that we use can be extended for the analysis of irregular LDPC codes as well.

1.4 Constructing of LDPC Codes

There are many ways to construct a parity-check matrix $\boldsymbol{H}$ or its Tanner graph for a given degree distribution pair. A common approach is to simply choose locations of non-zero elements in $\boldsymbol{H}$ randomly such that it conforms to the degree distribution pair [50]. The occurrence of 4-cycle loops can be prevented by ensuring that two columns can have at most one non-zero element at the same location.

Randomly constructed LDPC codes of large block lengths have proved to yield impressive performance [50, 9]. Unfortunately, random construction is usually not the best approach for LDPC codes of short block lenghts. In this work we construct instances of LDPC codes by means of the *progressive edge-growth (PEG) algorithm* [30]. It is a simple but efficient algorithm for constructing a Tanner graph with a large girth in a best-effort sense by consecutively establishing edges between variable and check nodes. A more detailed description of the algorithm and its performance can be found in [30].

1.5 Encoding LDPC Codes

One of the most serious problems concerning LDPC codes is their encoder complexity. In Section 1.2 it was shown how to calculate a codeword using a generator matrix $\boldsymbol{G}$. Unfortunately, if $\boldsymbol{H}$ is sparse, $\boldsymbol{G}$ is generally dense, meaning that it contains a significant number of non-zero elements. As a consequence, a naive encoder implementation has complexity quadratic in block length. Richardson *et. al* show that LDPC codes can be encoded with linear complexity if $\boldsymbol{H}$ is brought to an approximate lower triangular form [62].

1.6 Decoding LDPC Codes

In this section we analyze the decoding of an LDPC code defined by its parity-check matrix $\boldsymbol{H}$ over the additive white Gaussian noise (AWGN) channel. The transmitter uses binary phase shift keying (BPSK) to modulate each codeword bit c_i according to

$$x_i = 1 - 2c_i. \tag{15}$$

The modulated codeword $\boldsymbol{x} = [x_1, \ldots, x_N]$, $x_i \in \{-1, 1\}$ is transmitted over the AWGN channel and the receiver observes

$$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{n}, \tag{16}$$

where $\boldsymbol{n} = [n_1, \ldots, n_N]$ is the noise vector composed of N independent zero-mean Gaussian random variables n_i with variance σ^2. Note that $\boldsymbol{y} = [y_1, \ldots, y_N]$ and $y_i \in \mathbb{R}$. The receiver's task is to deduce which codeword $\boldsymbol{c}$ was transmitted.

1.6.1 Probability Domain Decoding

We focus on the decoding of the i-th bit c_i of the codeword $\boldsymbol{c}$. Much like optimal maximum a posteriori (MAP) decoding of trellis codes, we compute the a posteriori probability (APP) that c_i equals 1 given the received sequence $\boldsymbol{y}$ and under condition that all constraints upon c_i are satisfied. Let us begin by considering the APP ratio

$$\frac{\Pr[c_i = 0|\boldsymbol{y}, S_i]}{\Pr[c_i = 1|\boldsymbol{y}, S_i]},$$

where S_i is the event that occurs when bits in the hard decision word $\bar{\boldsymbol{c}}$, where $\bar{c}_i = 0$ if $y_i > 0$ and $\bar{c}_i = 1$ otherwise, satisfy the d_c parity-check constraints involving c_i. By applying

Bayes' formula, we get

$$\begin{aligned}\frac{\Pr[c_i = 0|\boldsymbol{y}, S_i]}{\Pr[c_i = 1|\boldsymbol{y}, S_i]} &= \frac{\Pr[c_i = 0|y_i]}{\Pr[c_i = 1|y_i]} \cdot \frac{\Pr[S_i|c_i = 0, \boldsymbol{y}]/\Pr[S_i]}{\Pr[S_i|c_i = 1, \boldsymbol{y}]/\Pr[S_i]} \\ &= \frac{\Pr[c_i = 0|y_i]}{\Pr[c_i = 1|y_i]} \cdot \frac{\Pr[S_i|c_i = 0, \boldsymbol{y}]}{\Pr[S_i|c_i = 1, \boldsymbol{y}]}, \qquad (17)\end{aligned}$$

Note that if $\bar{c_i} = 0$, the remaining $d_c - 1$ bits in each of the d_v parity-check constraints involving c_i must contain an even number of 1s for S_i to occur. On the other hand, if $\bar{c_i} = 1$, each parity-check constraint involving c_i must contain an odd number of 1s. The following Lemma will be helpful for further analysis.

Lemma 1.1. *Let $\boldsymbol{a} = [a_1, \ldots, a_m]$ be a sequence of m independent bits with $\Pr[a_i = 1] = P_i$. The probability that $\boldsymbol{a}$ contains an even number of 1s is*

$$\frac{1}{2} + \frac{1}{2}\prod_{i=1}^{m}(1 - 2P_i). \qquad (18)$$

Proof. We prove this by induction on m. If $\boldsymbol{a}$ contains an even number of 1s, the modulo-2 sum A_m of all bits in $\boldsymbol{a}$ is 0. For $m = 2$ we have

$$\begin{aligned}\Pr[A_2 = 0] &= \Pr[a_1 + a_2 = 0] \\ &= P_1P_2 + (1 - P_1)(1 - P_2) \\ &= \frac{1}{2} + \frac{1}{2}(1 - 2P_1)(1 - 2P_2) = \frac{1}{2} + \frac{1}{2}\prod_{i=1}^{2}(1 - 2P_i).\end{aligned}$$

For $m > 2$ assume that

$$\Pr[A_m = 0] = \frac{1}{2} + \frac{1}{2}\prod_{i=1}^{m}(1 - 2P_i)$$

holds; then, for $m + 1$ we get

$$\begin{aligned}\Pr[A_{m+1} = 0] &= \Pr[A_m + a_{m+1} = 0] \\ &= \frac{1}{2} + \frac{1}{2}(1 - 2\Pr[A_m = 1])(1 - 2P_{m+1}) \\ &= \frac{1}{2} + \frac{1}{2}\Big(1 - 2(1 - \Pr[A_m = 0])\Big)(1 - 2P_{m+1}) \\ &= \frac{1}{2} + \frac{1}{2}\prod_{i=1}^{m+1}(1 - 2P_i).\end{aligned}$$

□

Suppose we know that c_i is 0. A parity-check constraint involving c_i will be satisfied if among the remaining bits involved in that constraint there is an even number of 1s. On the other hand, if c_i is 1, there must be an odd number of 1s among the remaining bits. Now consider all bits that take part in the parity-check constraints involving c_i, assuming they are statistically independent. Then the probability that all parity-check constraints involving c_i are satisfied is the product of probabilities that each parity-check constraint is satisfied. With this fact Eq. (17) is expanded

$$\frac{\Pr[c_i = 0|\boldsymbol{y}, S_i]}{\Pr[c_i = 1|\boldsymbol{y}, S_i]} = \frac{\Pr[c_i = 0|y_i]}{\Pr[c_i = 1|y_i]} \cdot \frac{\prod_{j \in C_i} \frac{1}{2}\Big(1 + \prod_{k \in R_j \setminus \{i\}} (1 - 2\Pr[c_k = 1|y_k])\Big)}{\prod_{j \in C_i} \frac{1}{2}\Big(1 - \prod_{k \in R_j \setminus \{i\}} (1 - 2\Pr[c_k = 1|y_k])\Big)}, \tag{19}$$

where $R_j = \{i : H_{ji} = 1\}$ is a set of indices of columns with non-zero elements in the j-th row of $\boldsymbol{H}$, and $C_i = \{j : H_{ji} = 1\}$ is a set of indices of rows with non-zero elements in the i-th column of $\boldsymbol{H}$.

The probabilities in the first fraction on the right-hand side of Eq. (19) depend solely on y_i and are thus designated as *channel information.* In case of the AWGN channel and assuming a uniform source we get

$$\begin{aligned} \Pr[c_i = 0|y_i] = \Pr[x_i = 1|y_i] &= \frac{p(y_i|x_i = 1)\Pr[x_i = 1]}{p(y_i)} \\ &= \frac{\frac{1}{2}e^{-(y_i-1)^2/2\sigma^2}}{\frac{1}{2}e^{-(y_i-1)^2/2\sigma^2} + \frac{1}{2}e^{-(y_i+1)^2/2\sigma^2}} \\ &= \frac{1}{1 + e^{-2y_i/\sigma^2}} \end{aligned} \tag{20}$$

Now we can formulate an iterative algorithm for decoding LDPC codes, as originally proposed by Gallager [17] and known as the *message passing* algorithm. During execution, messages are iteratively exchanged between the neighboring nodes in the Tanner graph and each message is associated to the codeword bit corresponding to the variable node incident to the edge that carries the message. Effectively, the message conveys an estimate of that bit's most likely value along with some information about the estimate's reliability.

A message sent from either a check or a variable node along an edge should not depend on the message previously received along that edge. We say that only *extrinsic* information is passed along, which is an important property of good iterative decoders. A message from the variable node i to the check node j in the l-th iteration that carries the probability that the i-th bit has value k, is denoted by $q_{ij}^{(l)}(k)$. On the other hand, a message in the reverse direction from the check node j to the variable node i in the l-th iteration that carries the probability that the i-th bit has value k, is $r_{ji}^{(l)}(k)$.

Initially, the variable nodes only have information about the channel output values of their corresponding bits. Since no additional information from the neighboring check nodes is available, they send the message

$$q_{ij}^{(0)}(1) = \frac{1}{1 + e^{-\frac{2y_i}{\sigma^2}}} \tag{21}$$

along adjacent edges, in accordance with Eq. (20). In this case, the message gives the probability that the bit is 1, but it could also carry the probability that that bit is 0.

Subsequently, the messages are iteratively exchanged between check nodes and variable nodes. In each iteration, each check node receives messages from its neighboring variable nodes, processes that information, and passes the updated message

$$r_{ji}^{(l)}(1) = \frac{1}{2} - \frac{1}{2} \prod_{k \in R_j \setminus i} \left(1 - 2q_{kj}^{(l-1)}(1)\right) \tag{22}$$

back to the neighboring variable nodes, as illustrated in Figure 3. Similarly, each variable

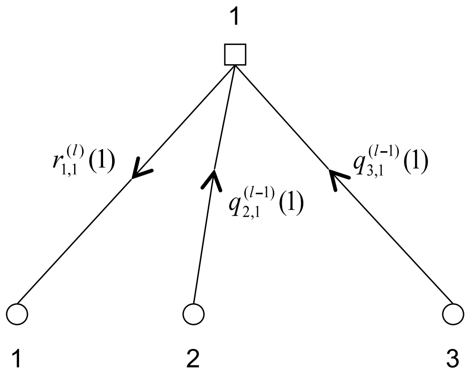

Figure 3: Check node message update.

node collects messages from its neighboring check nodes, calculates the probability that the corresponding bit is 1 and sends it to the neighboring check nodes. According to Eq. (19)

$$\frac{1-q_{ij}^{(l)}(1)}{q_{ij}^{(l)}(1)} = \frac{1-q_{ij}^{(0)}(1)}{q_{ij}^{(0)}(1)} \frac{\prod\limits_{k\in C_i\backslash j}\left(1-r_{ki}^{(l)}(1)\right)}{\prod\limits_{k\in C_i\backslash j} r_{ki}^{(l)}(1)}, \tag{23}$$

where the message received from the check node j was left out, since the updated message depends solely on extrinsic information, as in Eq. (22). From Eq. (23) we get

$$q_{ij}^{(l)}(1) = \frac{1}{1+\frac{1-q_{ij}^{(0)}(1)}{q_{ij}^{(0)}(1)}\frac{\prod_{k\in C_i\backslash j}\left(1-r_{ki}^{(l)}(1)\right)}{\prod_{k\in C_i\backslash j} r_{ki}^{(l)}(1)}}. \tag{24}$$

Figure 4 shows the message passing as seen from the variable node's perspective. When

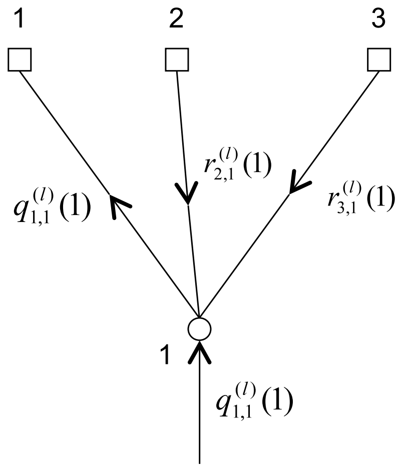

Figure 4: Variable node message update.

the variable node i receives messages from all neighboring check nodes, it calculates the probability, $\Pr[c_i = 1|\boldsymbol{y}, S_i]$ by taking into consideration *all* incoming check node messages like in Eq. (19). If $\boldsymbol{H}\hat{\boldsymbol{c}} = \boldsymbol{0}$, where $\hat{\boldsymbol{c}} = [\hat{c}_1, \ldots, \hat{c}_N]$ and

$$\hat{c}_i = \begin{cases} 1, & \text{if } \Pr[c_i = 1|\boldsymbol{y}, S_i] > \frac{1}{2}, \\ 0, & \text{otherwise,} \end{cases}$$

or if the maximum number of iterations has been reached, the algorithm stops; otherwise, the next iteration is started.

With this algorithm Eq. (19) converges to the maximum APP only if the messages are statistically independent and that is the case only if the Tanner graph corresponding to $\boldsymbol{H}$

is cycle-free. Unfortunately, Tanner graphs of practical codes are usually not cycle-free and in such cases the algorithm gives an approximate solution for the APP, which still yields very good performance.

1.6.2 Log-likelihood Domain Decoding

Up to this point, the decoder analysis has been treated in the probability domain. In Eqs. (19), (24), and (22) there is a substantial number of multiplications, which tend to become numerically unstable and are harder to implement in hardware as compared to additions. In order to simplify those equations, we introduce the term

$$\lambda_{c_i} = \log \frac{\Pr[c_i = 0]}{\Pr[c_i = 1]}, \tag{25}$$

called the *log-likelihood ratio (LLR)*. The probability distribution for a binary random variable c_i is uniquely specified by λ_{c_i}. The sign of λ_{c_i} indicates the most likely value of c_i, while its absolute value is a measure of certainty for that decision. In example, if $\Pr[c_i = 0]$ tends to 1 then λ_{c_i} tends to ∞. On the other hand, if $\Pr[c_i = 0] = \Pr[c_i = 1]$ then $\lambda_{c_i} = 0$, indicating that the value of c_i is unpredictable. Both probabilities $\Pr[c_i = 0]$ and $\Pr[c_i = 1]$ can be expressed by λ_{c_i}:

$$\begin{aligned} \Pr[c_i = 0] &= \frac{e^{-\lambda_{c_i}}}{1 + e^{-\lambda_{c_i}}}, \\ \Pr[c_i = 1] &= \frac{1}{1 + e^{-\lambda_{c_i}}}. \end{aligned}$$

Lemma 1.2. *In the log-likelihood domain, the updated check node message in the l-th iteration is given by*

$$\lambda_{r_{ji}}^{(l)} = 2 \tanh^{-1} \prod_{k \in R_j \setminus i} \tanh\left(\frac{1}{2} \lambda_{q_{kj}}^{(l-1)}\right). \tag{26}$$

Proof. If we rearrange Eq. (22) as

$$1 - 2r_{ji}^{(l)}(1) = \prod_{k \in R_j \setminus i} \left(1 - 2q_{kj}^{(l-1)}(1)\right) \tag{27}$$

and use the equality

$$\tanh\left(\frac{1}{2}\lambda_x\right) = 1 - 2\Pr[x = 1],$$

we get

$$\lambda_{r_{ji}}^{(l)} = \log \frac{r_{ji}^{(l)}(0)}{r_{ji}^{(l)}(1)} = 2\tanh^{-1} \prod_{k \in R_j \setminus i} \tanh\left(\frac{1}{2}\lambda_{q_{kj}}^{(l-1)}\right).$$

□

The formula for a variable node message in the log-likelihood domain can be derived from Eqs. (25), (24), (26), in the following way

$$\begin{aligned} \lambda_{q_{ij}}^{(l)} &= \log \frac{q_{ij}^{(l)}(0)}{q_{ij}^{(l)}(1)} \\ &= \log \frac{\Pr[c_i = 0|y_i]}{\Pr[c_i = 1|y_i]} + \log \frac{\prod\limits_{k \in C_i \setminus j} r_{ki}^{(l)}(0)}{\prod\limits_{k \in C_i \setminus j} r_{ki}^{(l)}(1)} \\ &= \lambda_{c_i|y_i} + \sum_{k \in C_i \setminus j} \lambda_{r_{ki}}^{(l)}. \end{aligned} \tag{28}$$

The first term on the right hand side of Eq. (28) is the contribution from the i-th channel output, while the second term sums contributions from the neighboring check nodes, with the recipient of $\lambda_{q_{i,j}}^{(l)}$ excluded. For an AWGN channel with the noise variance σ^2, Eq. (20) can be used to express $\lambda_{c_i|y_i}$ as

$$\lambda_{c_i|y_i} = \frac{2}{\sigma^2} y_i. \tag{29}$$

The constant of proportionality $2/\sigma^2$ is called the *channel reliability.*

The LLR for the bit corresponding to the variable node i after the l-th iteration is given by

$$\lambda_{c_i}^{(l)} = \lambda_{c_i|y_i} + \sum_{k \in C_i} \lambda_{r_{ki}}^{(l)}. \tag{30}$$

Here, as opposed to Eq. (28), all incoming check node messages are taken into account. After each iteration, the decoder evaluates the LLR values for each variable node and checks all parity-check constraints by verifying the relation $\boldsymbol{H}\hat{\boldsymbol{c}}^{(l)} = \mathbf{0}$, where $\hat{\boldsymbol{c}}^{(l)} = [\hat{c}_1^{(l)}, \ldots, \hat{c}_N^{(l)}]$ and

$$\hat{c_i}^{(l)} = \begin{cases} 1, & \text{if } \lambda_{c_i}^{(l)} < 0, \\ 0, & \text{otherwise.} \end{cases}$$

If all parity-check constraints are fulfilled or if the maximum number of iterations has been reached, the decoder stops; otherwise, the next iteration is started.

The following is a summary of

The Message-Passing Algorithm

Step 0.0 [**Initialization**] $\lambda_{r_{ji}}^{(0)} = 0$ for all $j \in \{1, \ldots, M\}$ and $i \in R_j$, $\lambda_{c_i|y_i} = \lambda_{c_i}^{(0)} = (2/\sigma^2)y_i$ for all $i \in \{1, \ldots, N\}$, $\lambda_{q_{ij}}^{(0)} = (2/\sigma^2)$ for all $i \in \{1, \ldots, N\}$ and $j \in C_i$, $l = 0$,

Step 1.0 [**Check Codeword**] If $\boldsymbol{H}\hat{\boldsymbol{c}}^{(l)} = \mathbf{0}$ or $l = l_{\max}$, then **STOP**, otherwise $l = l + 1$,

Step 1.1 [**Check Node Update**]

$$\lambda_{r_{ji}}^{(l)} = 2\tanh^{-1} \prod_{k \in R_j \setminus i} \tanh\left(\frac{1}{2}\lambda_{q_{kj}}^{(l-1)}\right)$$

for all $j \in \{1, \ldots, M\}$ and $i \in R_j$,

Step 1.2 [**Variable Node Update**]

$$\lambda_{q_{ij}}^{(l)} = \lambda_{c_i|y_i} + \sum_{k \in C_i \setminus j} \lambda_{r_{ki}}^{(l)}$$

and

$$\lambda_{c_i}^{(l)} = \lambda_{c_i|y_i} + \sum_{k \in C_i} \lambda_{r_{ki}}^{(l)}$$

for all $i \in \{1, \ldots, N\}$ and $j \in C_i$,

Step 2.0 [**Start New Loop**] go to Step 1.0.

CHAPTER II

PUNCTURED BINARY LDPC CODES

Modern communication systems operating over time variant channels, must be capable of adapting the rate according to the available channel state information in order to ensure the desired system performance and maximize the throughput.

Conventional applications usually employ a variety of fixed rate coding schemes that are well suited to the channel characteristics and meet the bit error rate (BER) requirements. Each of the coding schemes is optimized independently for the given rate to meet the desired performance. However, the complexity of such a system is rather high, since usually a separate encoder/decoder pair is required for each coding scheme.

An alternate way of realizing rate adaptability is to systematically puncture a low rate code, a so-called *mother code*, such that the bits of higher rate codes are a subset of the bits of lower rate codes. This approach was introduced by Hagenauer in [24], where puncturing was applied on convolutional codes. Its beauty lies in the fact that only one encoder/decoder pair is needed for the entire range of rates. Furthermore, a rate-compatible coding scheme enables the use of retransmission protocols based on incremental redundancy, such as the type-II hybrid Automatic Repeat Request (ARQ) protocol, which can increase the system throughput in presence of channel impairments. However, the choice of a coding scheme for different rates is restricted by compatibility constraints.

In this chapter we analyze punctured binary LDPC codes with block lengths of order 10^3; more specifically, we are interested in rate-compatible sets of punctured codes also referred to as rate-compatible punctured LDPC codes. We present ideas for puncturing regular binary LDPC codes proposed by Ha, Kim, and McLaughlin in [22], extend them for irregular binary LDPC codes, and introduce some new ideas that improve the performance at higher rates. Finally, we analyze the performance of punctured LDPC codes constructed according to the proposed ideas and compare them to constructions based on random puncturing.

2.1 Basic Ideas

Puncturing of the mother code can be performed either randomly or according to puncturing locations optimized for a given parity-check matrix. We refer to the former as *random puncturing* and to the latter as *intentional puncturing* in the subsequent sections. By smartly choosing sequences of bits to be punctured, we can significantly improve the code's performance at a wide range of rates as compared to random puncturing and we can avoid some selections that lead to catastrophic behavior to be discussed later. We often use Tanner graphs, where we refer to bits as variable nodes. Hence, a punctured variable node refers to the punctured bit corresponding to its variable node in the Tanner graph.

Suppose that some variable nodes were punctured. Assuming a uniform source, the decoder will set the LLR values of the punctured nodes to 0 at the initialization, since no a priori information is provided. Subsequently, the decoding process is started. If a punctured variable node receives a non-zero LLR message from one of its neighboring check nodes it is said to be *recovered*, regardless of whether the message implies the correct bit value or not. Through iterations all punctured variable nodes have to be recovered with the information provided by the unpunctured nodes, or else the parity-check constraints can not be satisfied.

Let $V = \{v_1, v_2, \ldots, v_N\}$ be the set of variable nodes and $P = \{p_1, p_2, \ldots, p_M\}$ the set of check nodes in the Tanner graph of the mother code. For the node x, we define its neighborhood $\mathcal{N}_x$ as the set of all nodes that can be reached from x by traversing a single edge; since the Tanner graph is bipartite, $\mathcal{N}_{v_i}$ is a subset of P, and $\mathcal{N}_{p_i}$ is a subset of V.

According to Eq. (30) a punctured variable node v will be recovered when at least one of the incoming check node messages is non-zero and by Eq. (26) the incoming check node message $\lambda_{r_{p,v}}$ will be non-zero, if all variable nodes in $\mathcal{N}_p \setminus \{v\}$ are either unpunctured or recovered. Accordingly, a punctured variable node v is referred to as *one-step recoverable* (1-SR) if v has at least one neighbor p, called the *survived check node*, such that all variable nodes in $\mathcal{N}_p \setminus \{v\}$ are unpunctured. The remaining check nodes from $\mathcal{N}_v$ are referred to as *dead check nodes* (see Figure 5).

In general, a punctured variable node v is called *k-step recoverable* (k-SR) if it has at least one neighbor p, called the survived check node, such that the set $\mathcal{N}_p \setminus \{v\}$ contains

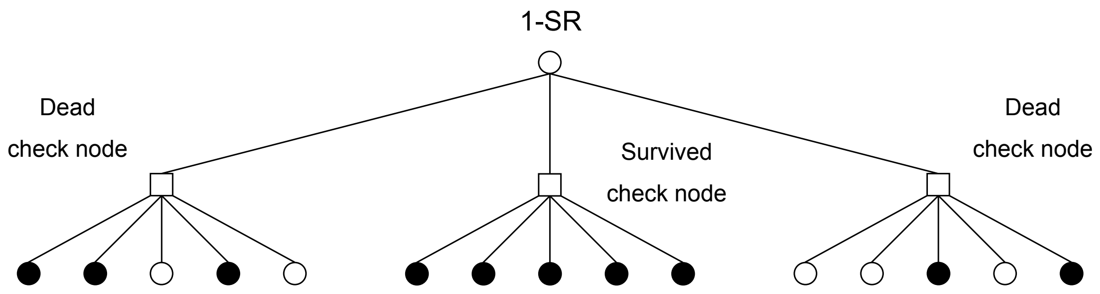

Figure 5: A 1-SR variable node. Void and filled circles are punctured and unpunctured variable nodes, respectively.

at least one $(k-1)$-SR node, while the rest are m-SR nodes, where $0 \leq m \leq k-1$.[1] Alternatively, we say that such variable node has *level of recoverability* k. Note that the *maximum level of recoverability* is defined as the highest level of recoverability of a punctured node. A k-SR node, $k \geq 1$, can have multiple survived check nodes, but the puncturing algorithm proposed in this section will ensure existence of at least one survived check node for each k-SR node. We refer to it as the *guaranteed survived check node*. Notice that a k-SR node will be recovered at the k-th iteration. An example of a 3-SR node is illustrated in Figure 6.

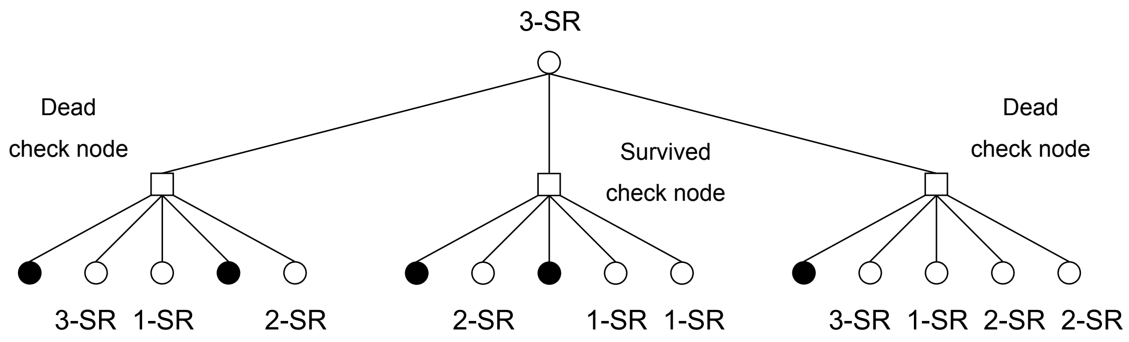

Figure 6: A 3-SR variable node. Void and filled circles denote punctured and unpunctured variable nodes, respectively.

The puncturing algorithm progressively determines variable nodes to be punctured and those to remain unpunctured. It is desirable that the punctured nodes be recovered in as few iterations as possible. Thus, the algorithm first attempts to maximize the number of

[1] A 0-SR node is an unpunctured node.

punctured nodes that are 1-SR. When no more 1-SR nodes can be found, it proceeds with 2-SR nodes, and so on, until every node in V has been chosen to be either unpunctured or punctured. The algorithm assigns the variable nodes to groups G_k, $k \geq 0$, where k indicates the level of recoverability of variable nodes in that group. For instance, if a variable node is chosen to be unpunctured, it is assigned to G_0, whereas if it is chosen to be punctured such that is will be recovered at the k-th iteration, it is assigned to G_k. At any particular stage during the execution of the algorithm, all variable nodes whose status (punctured or unpunctured) has not yet been determined are in the group G_∞. Hence, when the algorithm starts $G_\infty = V$, and when it stops G_∞ is empty.

Consider a k-SR node v. We build a tree originating from v in the following way: v is linked with its guaranteed survived check node c and subsequently c with all variable nodes from the set $\mathcal{N}_c \setminus \{v\}$; next, this process is repeated on every new punctured variable node in the tree until every branch terminates with an unpunctured variable node. The resulting tree is called the *recovery tree* of v. An example of a recovery tree is given in Figure 7. The number of unpunctured nodes in the recovery tree of v will be of importance; therefore, we denote it as $\mathcal{S}(v)$. If v is unpunctured, $\mathcal{S}(v)$ is set to 1.

Assume that v is recovered exclusively based on information received from the unpunctured nodes in its recovery tree. Then define the *recovery-error probability* of v, denoted $P_e(v)$, as the probability that v is recovered with a wrong value. Although a punctured node is not always recovered exclusively based on information from its recovery tree, $P_e(v)$ is helpful for selecting the best candidate for puncturing.

Some selections of punctured bits lead to a catastrophic behavior of the code. Namely, unless the punctured bits are selected carefully, it is possible that some of them are unrecoverable with the message-passing decoder. In other words, a fraction of punctured bits never receives a non-zero message, regardless of the number of iterations.

Consider a set $\mathscr{S}$ of variable nodes such that all their neighbors are connected to at least two variable nodes from $\mathscr{S}$. Such set is called a *stopping set* [14].

The punctured variable nodes that comprise a stopping set are unrecoverable [14]. The incoming messages from their neighboring check nodes are always zero—regardless of the

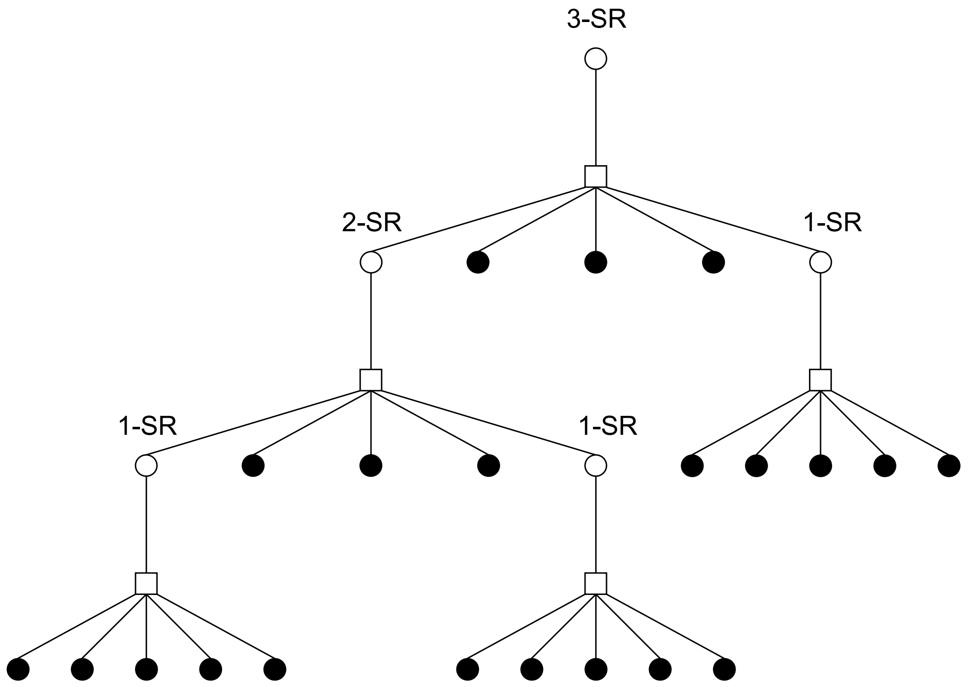

Figure 7: A recovery tree of a 3-SR node. Void and filled circles denote punctured and unpunctured variable nodes, respectively.

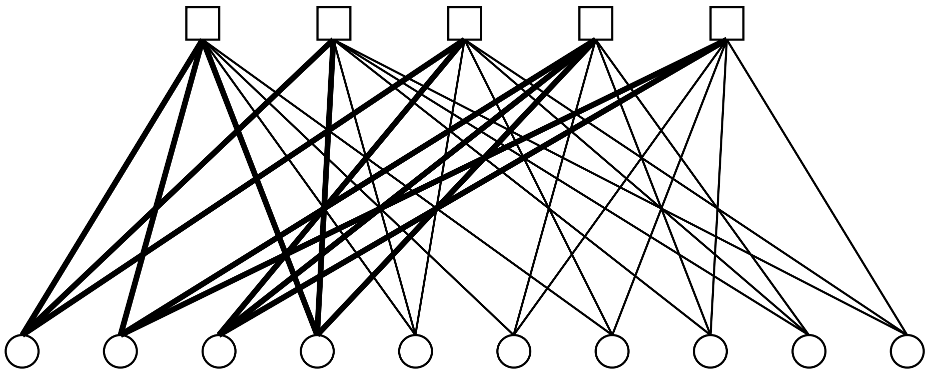

Figure 8: A stopping set. The four left most variable nodes form a stopping set.

number of iterations—since every neighboring check node is connected to at least two punctured variable nodes; therefore, puncturing a stopping set must be strictly avoided. In the following subsection we show how this can be done.

2.2 Criteria for Selecting Punctured Nodes

The main design goals here are to maximize the achievable rate and to ensure good performance at high rates, which is critical according to [20]. As we will see later in this chapter, these goals are often conflicting and call for a compromise. In this subsection we establish some guidelines for choosing punctured variable nodes that will yield a trade-off between the achievable rate and good performance at high rates.

Suppose we want to puncture a variable node $v \in G_\infty$ such that it is 1-SR. For v to be 1-SR there must exist at least one check node $p \in \mathcal{N}_v$ whose neighborhood $\mathcal{N}_p$ consists of variable nodes that belong to either G_0 or G_∞ (i.e., of unpunctured and unassigned variable nodes). We refer to all such check nodes as *candidate check nodes* and to their rows in $\boldsymbol{H}$ as *candidate rows*. If such check node p exists, v is assigned to G_1 and the remaining unassigned variable nodes in $\mathcal{N}_p$ are assigned to G_0; this guarantees that v will be recovered after the first iteration and p is designated as the *guaranteed survived check node of* v.

It is important to observe that assigning v to G_1 causes the size of G_0 to increase by the size of the set $\mathcal{N}_p \cap G_\infty$, designated as $|\mathcal{N}_p \cap G_\infty|$. This number will prove useful; therefore, we denote it as $\mathrm{rw}_{\mathrm{eff}}(p)$ and call it the *effective row weight* of p. Since we want to maximize the achievable code rate or equivalently, minimize the size of G_0, we choose the check node with the minimal effective row weight among the candidate check nodes. A similar approach can be applied to k-SR nodes.

Observe that if v has degree d, the number of check nodes that can serve as guaranteed survived check nodes to another 1-SR variable node is decreased by up to d. Namely, the d check nodes in $\mathcal{N}_v$ are already connected to 1-SR variable node v; therefore, no other 1-SR variable node in their neighborhoods can be recovered by them. It can happen that a check node is connected to several 1-SR (or k-SR) nodes, but is a guaranteed survived check node to none of them. Hence, suppose that a check node from $\mathcal{N}_v \setminus \{p\}$ serves as a guaranteed survived check node to another 1-SR node; then, there will remain only $d-1$ fewer candidate check nodes. As the size of G_1 is to be maximized, the decrease of candidate check nodes should be minimal. Towards this goal, we introduce *effective column weight* $\mathrm{cw}_{\mathrm{eff}}(v)$ of a variable node v, equal to the number of candidate check nodes in $\mathcal{N}_v$. The variable nodes with the lowest effective column weight are given priority, so that the decrease of candidate check nodes is minimal.

For further insight it is useful to focus on the recovery error probability P_e of a punctured variable node. In the following we show how P_e is calculated over the binary symmetric channel (BSC) and the AWGN channel.

Theorem 1. *Recovery error probability of a variable node $v \in G_k$ over the BSC with a crossover probability e is*

$$P_e(v) = \frac{1-(1-2e)^{\mathcal{S}(v)}}{2}. \tag{31}$$

Proof. We prove this by induction on k. For $k=0$ and $v \in G_0$, we have

$$P_e(v) = \frac{1-(1-2e)}{2} = e,$$

which is true, since v is unpunctured. Moreover, for $k=1$ and $v \in G_1$,

$$P_e(v) = \frac{1-(1-2e)^{d_p-1}}{2}, \tag{32}$$

since all remaining variable nodes in the recovery tree of v are in G_0 and therefore Eq. (18) applies. Assume that for $0 \le m \le k$ and any $v \in G_m$ Eq. (31) holds; then, for $k+1$ and $v \in G_{k+1}$,

$$P_e(v) = \frac{1-\prod_{j=1}^{d_p-1}\Big(1-2P_e(\gamma_j)\Big)}{2},$$

where γ_j are variable nodes from the set $\mathcal{N}_p \setminus \{v\}$ and p is the guaranteed survived check node of v. Since each γ_j belongs to one of the groups G_m, $0 \leq m \leq k$, we have

$$\begin{aligned}
P_e(v) &= \frac{1 - \prod_{j=1}^{d_p-1}\left(1 - 2\frac{1-(1-2e)^{\mathcal{S}(\gamma_j)}}{2}\right)}{2} \\
&= \frac{1 - \prod_{j=1}^{d_p-1}(1-2e)^{\mathcal{S}(\gamma_j)}}{2} \\
&= \frac{1 - (1-2e)^{\sum_{j=1}^{d_p-1}\mathcal{S}(\gamma_j)}}{2} \\
&= \frac{1 - (1-2e)^{\mathcal{S}(v)}}{2}.
\end{aligned}$$

□

Observe that the recovery error probability of a punctured variable node over the BSC grows with the number of unpunctured variable nodes in its recovery tree. In fact, the same holds for the AWGN channel, but before we prove it, we briefly summarize some earlier work.

The probability density function (PDF) of a check node message[2] can be approximated with a Gaussian distribution over the AWGN channel [10]. For output symmetric channels[3] the variance σ^2 and the mean of the check node message PDF m_u are related by $\sigma^2 = 2m_u$, which is preserved through iterations. Thus, if statistical properties of a check node message PDF through iterations are of interest, it is sufficient to follow the mean m_u, called the *updated mean*, of its Gaussian PDF. After any iteration the bit error probability can be calculated from the message PDF using the Q-function.[4]

On output symmetric channels the bit error probability of a message-passing decoder does not depend on the transmitted codeword [60]. Consequently, it can be assumed without any loss of generality that the all-zero codeword is transmitted. In such case, the growing updated mean results in a decrease of the bit error probability.

According to [10] the updated mean $m_u(v)$ is given by

$$m_u(v) = \phi^{-1}\left(1 - \prod_{i=1}^{d_p-1}\left(1 - \phi(m_{v_i})\right)\right), \tag{33}$$

[2]The messages are assumed to be in the log-likelihood domain.

[3]A channel is output symmetric if $p(y_i = q|x_i = 1) = p(y_i = -q|x_i = -1)$, where x_i is an input bit and y_i is the channel output of that bit.

[4]A more in-depth analysis of the variable and check node message PDFs can be found in [60, 61, 10].

where m_{v_i} is the mean of the Gaussian PDF of the message from a variable node incident to that check node,

$$\phi(x) = \begin{cases} 1 - \frac{1}{\sqrt{4\pi x}} \int_{\mathbb{R}} \tanh \frac{u}{2} e^{\frac{-(u-x)^2}{4x}} du, & \text{if } x > 0 \\ 1, & \text{if } x = 0, \end{cases}$$

and $\phi^{-1}(x)$ is the inverse function of $\phi(x)$.

Let m_{u_0} be the mean of the LLR of the channel output, let $Q(\cdot)$ be the Q-function.

Theorem 2. *Recovery error probability of a variable node $v \in G_k$ over the AWGN channel is given by*

$$P_e(v) = Q(\sqrt{m_u(v)/2}), \tag{34}$$

where

$$m_u(v) = \phi^{-1}\Big(1 - \Big[1 - \phi(m_{u_0})\Big]^{\mathcal{S}(v)}\Big). \tag{35}$$

Proof. The proof is by induction on k. For $k = 1$, the updated mean $m_u(v)$ of a message, which recovers a punctured variable node $v \in G_1$ is given by $\phi^{-1}(1 - [1 - \phi(m_{u_0})]^{d_p - 1}) = \phi^{-1}(1 - [1 - \phi(m_{u_0})]^{\mathcal{S}(v)})$, as $m_{\gamma_j} = m_{u_0}$ if $\gamma_j \in G_0$. Assume that for $1 \le i \le k$ and any $v \in G_i$, Eq. (35) holds. Then, for $k+1$ and $v \in G_{k+1}$

$$m_u(v) = \phi^{-1}\Big(1 - \prod_{j=1}^{d_p - 1} \Big[1 - \phi(m_{\gamma_j})\Big]\Big).$$

Since for all γ_j belong to G_i, where $0 \le i \le k$, and $m_{\gamma_j} = m_u(\gamma_j)$, we have

$$\begin{aligned} m_u(v) &= \phi^{-1}\Big(1 - \prod_{j=1}^{d_p-1} \Big[1 - \phi(m_{\gamma_j})\Big]\Big) \\ &= \phi^{-1}\Big(1 - \prod_{j=1}^{d_p-1} \Big[1 - \phi(m_{u_0})\Big]^{\mathcal{S}(\gamma_j)}\Big) \\ &= \phi^{-1}\Big(1 - [1 - \phi(m_{u_0})]^{\sum_{j=1}^{d_p-1} \mathcal{S}(\gamma_j)}\Big) \\ &= \phi^{-1}\Big(1 - \Big[1 - \phi(m_{u_0})\Big]^{\mathcal{S}(v)}\Big), \end{aligned}$$

where d_{p} is a degree of the guaranteed survived check node of v, γ_j are variable nodes in $\mathcal{N}_p \setminus \{v\}$, and $\gamma_j \in G_m$ for $0 \le m \le k$. □

The graph of the function $\phi(x)$ is shown in Figure 9. As $\mathcal{S}(v)$ increases, so does the argument to the ϕ^{-1} function in Eq. (35); in effect, the updated mean decreases, and the recovery error probability $P_e(v)$ increases. Consequently, in the process of choosing variable nodes to be punctured, those with lower $\mathcal{S}(v)$ are preferred. Since in most cases $\mathcal{S}(v)$ grows

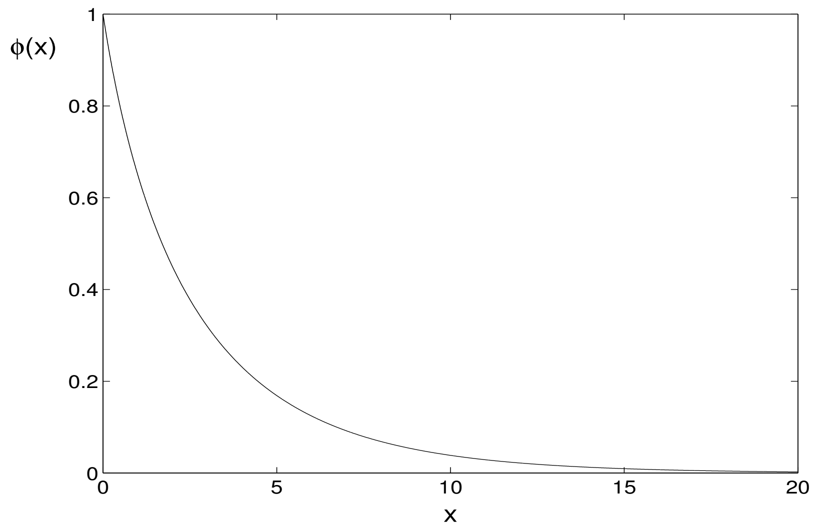

Figure 9: The plot of the function $\phi(x)$.

with the level of recoverability of v, the first goal is to maximize the size of G_1. When no more 1-SR nodes are found, the number of 2-SR nodes is maximized, and so on, until no more variable nodes can be punctured.

The algorithm based on this discussion, called the *grouping algorithm*, is introduced next.

2.3 The Grouping Algorithm

Having established the selection criteria for choosing the most appropriate variable node to be punctured, we start by presenting the grouping algorithm and afterwards we give a short explanation to each step. In each algorithm loop one variable node is chosen, which has the lowest effective row weight (to maximize G_0), the lowest effective column weight (to maximize the size of G_k), and the lowest recovery error probability.

The Grouping Algorithm

Step 0 [**Initialization**] For a given $M \times N$ parity-check matrix $\mathbf{H}$, set $k = 1$, $R_\infty = P = \{p_1, p_2, \ldots, p_M\}$, $G_0 = \emptyset$, $G_1 = \emptyset$, $G_\infty = V = \{v_1, v_2, \ldots, v_N\}$, $\mathcal{S}(v_i) = 1$ for all $1 \leq i \leq N$.

Step 1 [**Group Variable Nodes**] Form $G_\infty^p = \mathcal{N}_p \cap G_\infty$ for each $p \in R_\infty$.

Step 2 [**Find Check Nodes With Minimal Effective Row Weight**] Make a subset $R_{\infty,\min}$ of R_∞ such that for all $p \in R_{\infty,\min}$, $\mathrm{rw}_{\mathrm{eff}}^{\min} = \mathrm{rw}_{\mathrm{eff}}(p) \leq |G_\infty^p| = \mathrm{rw}_{\mathrm{eff}}(p')$ for every $p' \in R_\infty$.

Step 3.0 [**Group Check Nodes**] Make $R_\infty^v = \mathcal{N}_v \cap R_\infty$ for all $v \in G_\infty^p$, and $p \in R_{\infty,\min}$.

Step 3.1 [**Find Best Check Nodes**] Find a subset $R_{\infty,\min}^*$ of $R_{\infty,\min}$, such that for all $p^* \in R_{\infty,\min}^*$, there exists a $v \in G_\infty^{p^*}$ such that $|R_\infty^v| = \mathrm{cw}_{\mathrm{eff}}^{\min} = \mathrm{cw}_{\mathrm{eff}}(v) \leq |R_\infty^{v'}| = \mathrm{cw}_{\mathrm{eff}}(v')$ for every $v' \in R_\infty^p$ and $p \in R_{\infty,\min}$.

Step 3.2 [**Make a Set of Ordered Pairs**] Build a set of ordered pairs $\mathcal{O} = \{(p_1^*, v_1^*), (p_2^*, v_2^*), \ldots, (p_n^*, v_n^*)\}$, where p_i^* and v_i^* are check nodes and variable nodes with $\mathrm{cw}_{\mathrm{eff}}^{\min}$ and $\mathrm{rw}_{\mathrm{eff}}^{\min}$, respectively. If there are multiple ordered pairs with the same variable node $v^* \in G_\infty^{p^*}$, pick one of them randomly and remove the remaining ones, so that each variable node is represented at most once in $\mathcal{O}$.

Step 3.3 [**Find The Best Pair**] Let $\mathcal{W}(p^*, v^*) = \sum_{v \in \mathcal{N}_{p^*} \setminus \{v^*\}} \mathcal{S}(v)$. Pick a pair (p^*, v^*) from $\mathcal{O}$ such that $\mathcal{W}(p^*, v^*) \leq \mathcal{W}(p_i^*, v_i^*)$, for all $1 \leq i \leq n$. If there are multiple such pairs, pick one randomly,

Step 4 [**Update**] Set $G_k = G_k \cup \{v^*\}$, $G_0 = G_0 \cup \left(G_\infty^{p^*} \setminus \{v^*\}\right)$, $G_\infty = G_\infty \setminus G_\infty^{p^*}$, $R_\infty = R_\infty \setminus R_\infty^{v^*}$, $\mathcal{S}(v^*) = \mathcal{W}(p^*, v^*)$,

Step 5.0 [**Check Stop Condition**] If $|G_\infty| = 0$, then **STOP**,

Step 5.1 [**Decision**] If $|R_\infty| \geq 1$, go to Step 1, Step 5.2 [**Set New R_∞ and Increment** k] Set $k = k+1$, $R_\infty = \{p_i : \text{all } p_i, 1 \leq i \leq M,$ for which $\text{rw}_{\text{eff}}(p_i, G_\infty) > 0\}$, and go to Step 1.

Initialization with $k = 1$ indicates that the algorithm starts by puncturing 1-SR variable nodes. Sets R_∞ and G_∞ initially contain all check nodes and variable nodes, respectively. Throughout the algorithm R_∞ contains check nodes that have no neighbors in G_k and at least one in G_∞. In effect, R_∞ contains indices of candidate check nodes or rows. For all variable nodes $S(v)$ is set to 1, the value for unpunctured nodes, and will be updated for each unassigned node chosen for puncturing.

In step 1, sets G_∞^p are built for every $p \in R_\infty$, such that G_∞^p contains all unassigned variable nodes connected to the check node p. The effective row weight of p is determined by $|G_\infty^p|$. The algorithm searches for check nodes with the lowest row weight and stores them in $R_{\infty,\min}$. In the following steps, only unassigned variable nodes connected to $R_{\infty,\min}$ are considered as candidates for puncturing in order to ensure smallest possible increase of G_0.

Beside minimizing G_0, it is desired that G_k be maximized and that the chosen punctured node have minimal recovery error probability. The algorithm reduces the number of candidate variable nodes by choosing those with the lowest effective column weight in steps 3.0 and 3.1; this will cause the number of candidate check nodes (or the size of R_∞) to decrease minimally. In step 3.0 the set R_∞^v is built for each candidate variable node v, containing check nodes connected to v and represented in $R_{\infty,\min}$. The effective column weight of v equals $|R_\infty^v|$. Check nodes that are connected to candidate variable nodes with the lowest effective column weight form the set $R_{\infty,\min}^*$.

To organize the remaining candidate variable nodes the set $\mathcal{O}$ of ordered pairs (p^*, v^*) is built, where p^* is the check node with the lowest effective row weight among check nodes in $\mathcal{N}_{v^*}$ and v^* is the candidate variable node with the lowest effective column weight. It may happen that v^* is connected to more than one check node with the lowest effective row weight; in such case, one of those check nodes is chosen randomly ensuring that no two ordered pairs in $\mathcal{O}$ share a v^*.

In step 3.3, the candidate variable node v^* from the ordered pair (p^*, v^*) with the minimal number of unpunctured nodes in its recovery tree, here denoted by $\mathcal{W}(p^*, v^*)$, is chosen for puncturing, while its guaranteed survived check node is p^*. If there are multiple candidate variable nodes with the lowest $\mathcal{W}(p^*, v^*)$, one of them is chosen randomly.

Subsequently, in step 4 the algorithm updates some sets and variables to ensure correct operation in the next loop: the chosen variable node v^* is added to G_k while the remaining unassigned variable nodes connected to p^* are set to remain unpunctured and assigned to G_0. All previously unassigned variable nodes connected to p are removed from G_∞, and the number of unpunctured nodes in the recovery tree $\mathcal{S}(v^*)$ is set to $\mathcal{W}(p^*, v^*)$.

In steps 5.0–5.2 it is checked if there are any remaining unassigned variable nodes. If none is found, no additional variable nodes can be punctured and the algorithm stops; otherwise, a search for additional k-SR nodes is performed by checking the number of elements in R_∞. If R_∞ contains at least one check node, additional k-SR variable nodes can be found, and a new loop is started without incrementing k. In the opposite case, k is incremented by one and the algorithm starts searching for $(k+1)$-SR nodes. Before the new loop is started, a new set R_∞ is built to ensure the correct calculation of the effective column weight in the subsequent loops.

When the algorithm stops, every variable node belongs to one of the sets (also called groups) G_i, $0 \leq i \leq K$, where K designates the maximum level of recoverability. These groups form a set

$$\boldsymbol{G} = \{G_0, G_1, \ldots, G_K\}.$$

The maximum achievable rate $R_{\max}$ is now expressed as

$$R_{\max} = \frac{R_0}{1 - \sum_{i=1}^{K} |G_i|/N}, \tag{36}$$

where R_0 is the rate of the mother code and N is the block length. For an increasing sequence of desired rates, $R_0 \leq R_1 \leq \ldots \leq R_L \leq R_{\max}$, let P_i be the set of variable nodes that are punctured to achieve rate R_i. The P_i is required to contain

$$|P_i| = \left\lfloor \frac{N(R_i - R_0)}{R_i} \right\rfloor \tag{37}$$

variable nodes. To achieve $R_{\max}$ variable nodes are punctured as determined by the grouping algorithm, thus $P_{\max} = \boldsymbol{G} \setminus \{G_0\}$; however, to achieve rates between R_0 and $R_{\max}$ only a fraction of the nodes in $\boldsymbol{G} \setminus \{G_0\}$ has to be punctured. Note that P_i must be a subset of P_j if $R_i < R_j$ for the codes are to be rate-compatible. The question is, how the sets $P_1, P_2, \ldots, P_L$ should be determined.

Intuitively, we decide to puncture variable nodes from the lower indexed groups first. If only a fraction of nodes from a certain group should be punctured to achieve a given rate, we give priority to those with most survived check nodes. For example, suppose that $\boldsymbol{G}$ was built such that $|G_0| = 600$, $|G_1| = 200$, $|G_2| = 100$, $|G_3| = 50$ and $|G_4| = 20$, and we require sets P_1 and P_2 such that $|P_1| = 250$ and $|P_2| = 310$. Then 250 variable nodes have to be punctured to achieve R_1. We start by puncturing variable nodes in G_1 and since $|G_1| < |P_1|$, P_1 will contain all variable nodes from G_1. The remaining 50 are chosen from G_2 with priority given to the variable nodes with a higher number of survived check nodes. When determining the number of survived check nodes, we assume that variable nodes in G_3 and G_4 are in G_0, as they will not be punctured to achieve R_1. Similarly, to achieve R_2 we puncture all nodes from P_1, the variable nodes from G_2 that are not in P_1, and 10 nodes from G_3.

2.4 Simulation Results

In this section we discuss the performance of LDPC codes when they are punctured using two approaches: intentionally, according to the grouping algorithm, and randomly. We consider irregular punctured LDPC codes with a mother code of rate 0.5 and the comparisons between the two approaches will be drawn at block lengths 1024 and 4096.

Each mother code is constructed with the PEG algorithm using a variable node degree distribution from [20]

$$\lambda(x) = 0.25105x + 0.30938x^2 + 0.00104x^3 + 0.43853x^9.$$

Subsequently, each mother code is punctured to achieve rates 0.6, 0.7, and 0.8. First, we analyze the group distribution of the randomly chosen punctured nodes and compare it to the group distribution of the intentionally punctured nodes. The Tables 1 and 2 summarize the results for block lengths $N = 1024$ and $N = 4096$, respectively, where the distributions for achieving the rate 0.8 are shown.

Table 1: Group distribution of the variable nodes, block length $N = 1024$.

Group	Intentional	Random	Group	Intentional	Random
G_0	640	640	G_7	0	32
G_1	315	70	G_8	0	20
G_2	62	58	G_9	0	14
G_3	7	42	G_{10}	0	10
G_4	0	43	G_{11}	0	4
G_5	0	38	G_{12}	0	2
G_6	0	51	G_{13}	0	0

The punctured variable nodes obtained by intentional puncturing require considerably less iterations to recover than those obtained by random puncturing. While 3 iterations suffice to recover all punctured variables nodes for intentional puncturing, random puncturing requires 12, or 4 times as many. Moreover, if the mother is code is punctured intentionally, most of the punctured nodes are 1-SR—almost 80 %. On the other hand 80 % of the punctured nodes are almost uniformly distributed in groups from G_1 to G_6 when punctured randomly. Since the higher level of recoverability usually results in a higher recovery error probability, the intentionally punctured LDPC code is expected to perform better.

In Figure 10 the performance of intentional and random puncturing of the mother code with a block length of 1024 is compared over the AWGN channel. At the rate 0.6, the number of punctured variable nodes is 170 and intentional puncturing performs better with a margin of 0.15 dB at the BER of 10^{-5}. As the rate increases the performance gap between intentional and random puncturing grows steadily. At the rate 0.8 the number of punctured variable nodes is 384 and the gap reaches 1 dB at the BER of 10^{-5}.

Next, we evaluate the performance at block length 4096, shown in Figure 11. Again, the performance gap between intentional and random puncturing steadily grows with the increasing rate. At the rate 0.6 the gap is smaller than 0.1 dB at the BER of 10^{-5}, at 0.7 it rises to 0.2 dB, and at the highest rate of 0.8, where the number of punctured variable nodes is 1536, the gap between intentional and random puncturing reaches 0.9 dB.

Another important aspect is the number of iterations required for successful decoding.

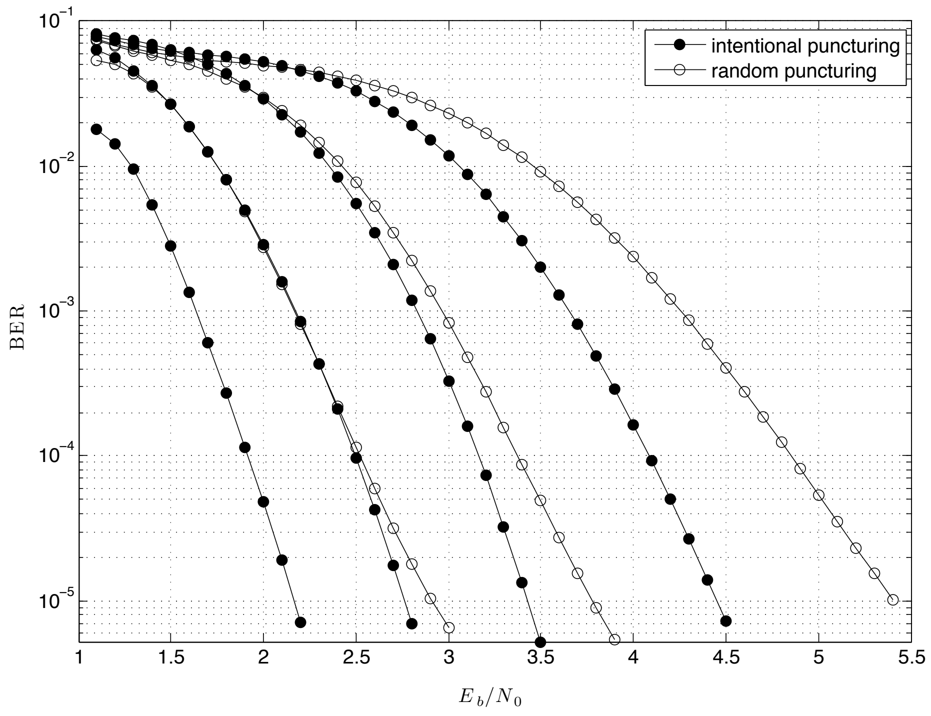

Figure 10: Comparison between the intentionally and randomly punctured LDPC code; the mother code has a block length of 1024. The curves from left to right correspond to: the unpunctured mother code and the punctured codes at rates 0.6, 0.7, and 0.8.

Table 2: Group distribution of the variable nodes, block length $N = 4096$.

Group	Intentional	Random	Group	Intentional	Random
G_0	2560	2560	G_7	0	146
G_1	1209	221	G_8	0	94
G_2	292	200	G_9	0	58
G_3	35	195	G_{10}	0	25
G_4	0	210	G_{11}	0	5
G_5	0	192	G_{12}	0	1
G_6	0	189	G_{13}	0	0

Inspection of the group distribution of punctured variable nodes reveals how many iterations are required for all punctured variable nodes to be recovered. However, a punctured variable node may be recovered with a wrong value and then additional iterations will be required to correct the error. In Figure 12 the distribution of required numbers of iterations for intentional and random puncturing at the rate 0.8 and BER of 10^{-5} is shown, where we observed the fraction of codewords (percentage-wise) successfully decoded with respect to

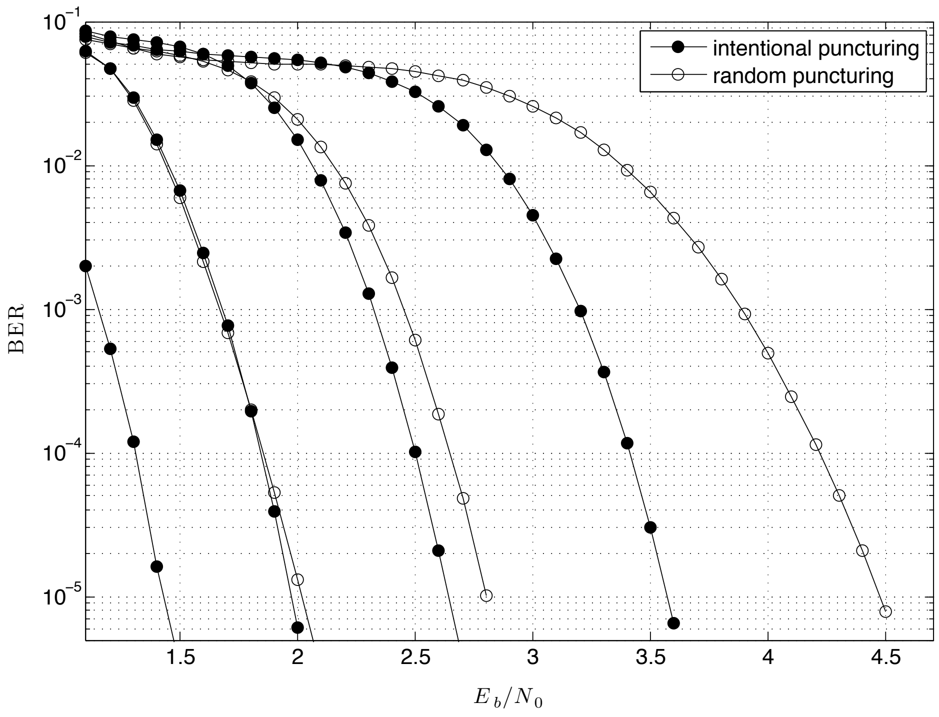

Figure 11: Comparison between the intentionally and randomly punctured LDPC code; the mother code has a block length of 4096. The curves from left to right correspond to: the unpunctured mother code and the punctured codes at rates 0.6, 0.7, and 0.8.

the number of iterations. The results for intentional puncturing and random puncturing were observed at $E_b/N_0 = 3.6$ dB and $E_b/N_0 = 4.5$ dB (both at BER $= 10^{-5}$), respectively. LDPC codes obtained by random puncturing require 3–4 more iterations on average for successful decoding.

When applying the algorithm, random selections are made at various stages if there are multiple best candidates. Accordingly, the final group distribution generally depends on the random seed used to initialize the algorithm; however, in our tests we saw only small differences. It is interesting to check the highest achievable rate when puncturing is performed according to the grouping algorithm. By and large, the lower the size of G_0, the higher the achievable rate. The algorithm was applied to the mother code of block length 4096 multiple times with different random seeds and the lowest size of G_0 that we obtained was 2500. Hence, the maximum number of punctured nodes was 1596, which results in the highest achievable rate of 0.82.

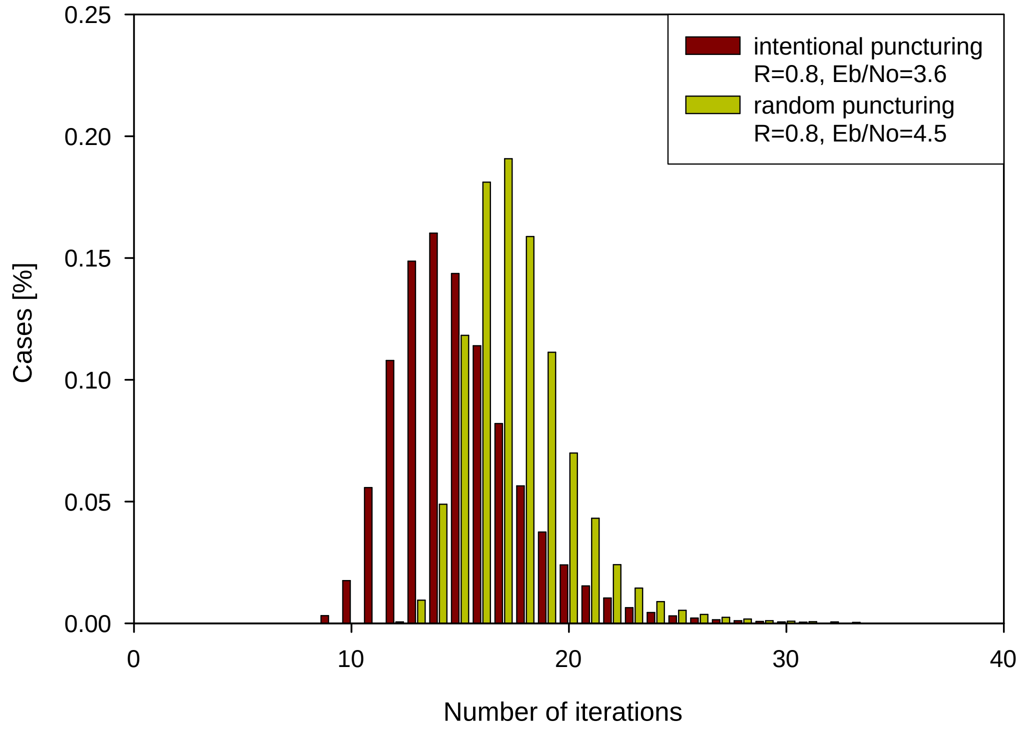

Figure 12: The distribution of required number of iterations for intentional and random puncturing over the AWGN channel at $R = 0.8$ and BER $= 10^{-5}$.

It is useful to note that the highest achievable rate can be increased by limiting the size of groups G_k, $k \geq 1$ during the execution of the algorithm. In the extreme case, the size of each group G_k, $k \geq 1$, can be limited to 1 and achievable rate will reach the maximum. Unfortunately, the performance at higher rates is degraded significantly due to higher levels of recoverability of punctured variable nodes.

2.5 Layered BP Decoding for Punctured LDPC Codes

The previous section revealed that punctured LDPC codes generally require more decoding iterations that unpunctured codes. In actual applications this can pose a serious problem as slower convergence results in higher power consumption and longer latency. In this section we show how convergence of punctured LDPC codes can be significantly accelerated using a carefully designed layering scheme for the layered BP algorithm proposed in [29, 74, 36].

The layered BP algorithm is a modification of the conventional BP algorithm described in Section 1.6.2, where the check nodes are divided in subgroups called *layers* and each

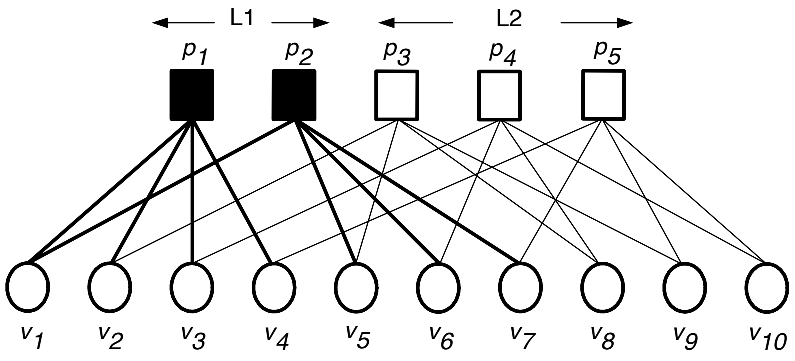

Figure 13: A Tanner graph split into two check nodes layers: L_1 and L_2.

iteration is broken into multiple *subiterations.* More specifically, the layered BP decoding divides the Tanner graph of an LDPC code into subgraphs, such that each subgraph consists of a set of check nodes and all their neighboring variable nodes. Each check node appears in exactly one subgraph, while variable nodes may appear in multiple subgraphs. The conventional decoding iteration is split into multiple subiterations, such that in each subiteration the check node and variable node updates are calculated in one subgraph. The decoding then progresses sequentially through subgraphs by performing message updates subiteration by subiteration. A parity-check test over the entire codeword is performed at the end of each subiteration. Figures 13 and 14 show an example of a Tanner graph split into subgraphs. Layered BP algorithm yields faster decoding convergence but it does not improve the BER when the maximum number of iterations is large enough. In practical communication systems the maximum number of iterations is limited and faster decoding convergence results in better BER performance.

In [36], “random” and “bimodal” layering of check nodes is studied for unpunctured regular LDPC codes. The performance differences between the layerings were shown to be only marginal. On the other hand, when an LDPC code is punctured, significant performance gains are achievable by a careful layering selection. In the following, we explain how a good layering structure can be obtained.

We propose an approach where check nodes are layered based on levels of recoverability of neighboring puncturing variables nodes as follows. If a check node is a survived check node to a k-SR punctured variable node, it is assigned to the layer L_k, where $1 \leq k \leq K$.

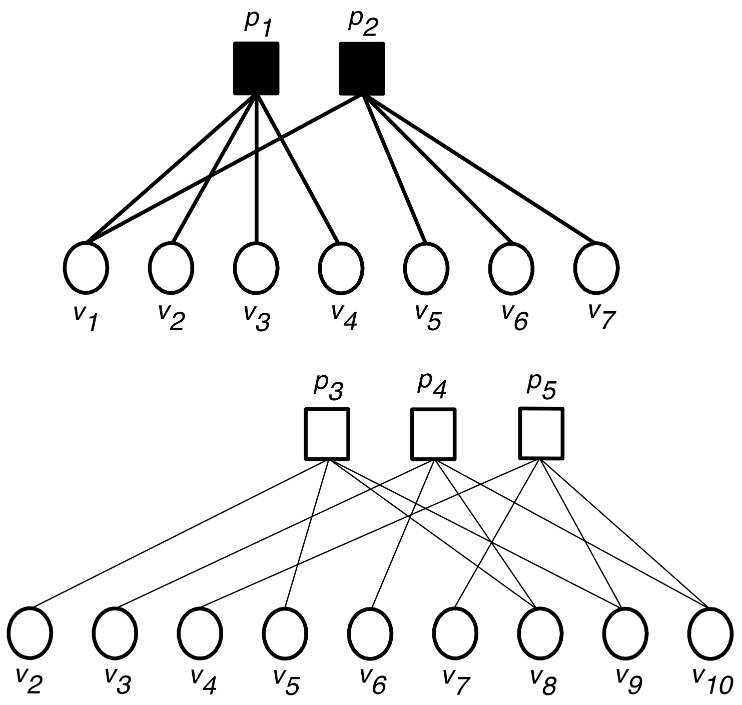

Figure 14: Layers L_1 and L_2 of the Tanner graph in Figure 13 and their subgraphs.

By upper bounding k with K we assume that the maximum level of recoverability is K. All remaining check nodes, i.e. those that do not recover any punctured variable nodes, are assigned to the last layer L_{K+1}. When a high percentage of variable nodes are punctured, the layer L_{K+1} may not contain any check nodes.

The example in Figure 15 shows how a 3-SR node is recovered under layered BP decoding with the proposed layering of check nodes. In the first subiteration the check nodes p_2 and p_3 (both in L_1) recover 1-SR variable nodes v_{10} and v_{13}; in the second subiteration the check node p_4 recovers 2-SR variable node v_{14}; now all neighbors except v_{15} of p_5 have non-zero LLRs, and in the third subiteration v_{15} will be recovered by p_5.

If the layers from L_1 to L_{K+1} are processed in ascending order, all punctured variable nodes are recovered in the first iteration. Hence, it is ensured that the punctured variable nodes get involved in the decoding process very quickly, which results in faster convergence.

What follows is an algorithm for determining the layer for each of M check nodes.

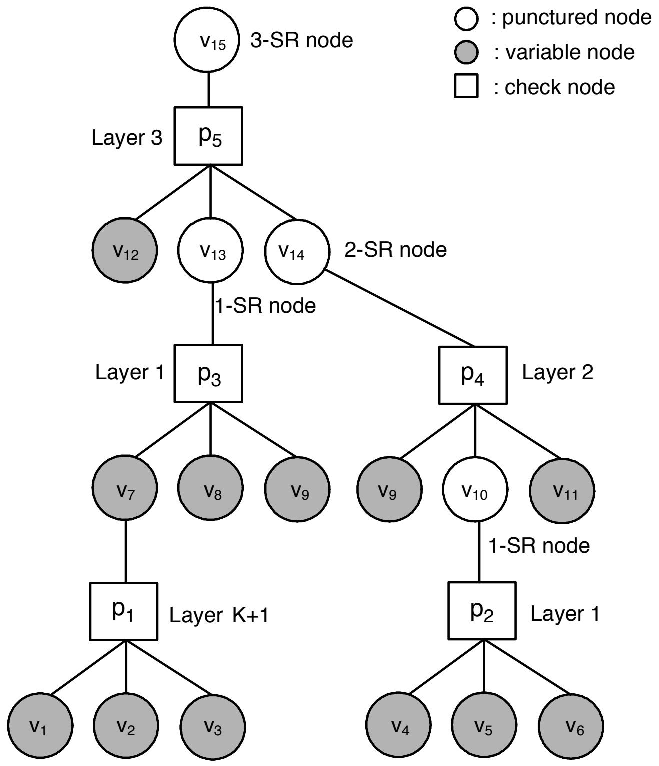

Figure 15: Recovery tree of a 3-SR node where $p_2, p_3 \in L_1$, $p_4 \in L_2$, $p_5 \in L_3$, $p_1 \in L_{K+1}$, and $K \geq 3$.

[**Loop**] For each check node $p \in P$:

Step 1 [**Find maximum level of recoverability**] Find the highest level of recoverability among variables nodes in $\mathcal{N}_p$ and denote it is k.

Step 2 [**Assign layer**] If $1 \leq k \leq K$, assign p to layer L_k, otherwise assign it to layer L_{K+1}.

Note that the layering is performed based on levels of recoverability of punctured variable nodes at the highest considered code rate. For all intermediate code rates, the layering remains unchanged.

To verify the effectiveness of the this layering scheme we construct an LDPC mother code with the degree distribution pair (from [10])

$$\lambda(x) = 0.30780x + 0.27287x^2 + 0.41933x^6 \text{ and}$$

$$\rho(x) = 0.4x^5 + 0.6x^6$$

with a code rate 0.5 and block length 2000. The mother code is punctured using the previously described grouping algorithm to obtain code rates 0.6, 0.7, 0.8, and 0.9. Information about the obtained check node layering is shown in Table 3. Notice that the highest level of recoverability among the punctured variable nodes is 5 and each check node serves as a survived check node to one punctured variable node, hence there is no layer L_6.

Table 3: The distribution of check nodes in layers. The highest level of recoverability among the punctured variable nodes is 5.

	Proposed	Random1	Random2
Layer 1	222	200	222
Layer 2	244	200	244
Layer 3	178	200	178
Layer 4	178	200	178
Layer 5	178	200	178

We compare the proposed layering with random layering. In one case we create random layering with uniform layer sizes and in the other, the layer sizes we match those from the proposed layering. The performance of these two random layerings is virtually indistinguishable; therefore, they are represented by a single curve in the following figures.

Figure 16 shows the simulation results over an AWGN channel with the maximum number of iterations set to 15. The performance improvement of the proposed over random layering increases with the growing code rate. This behavior is to be expected, for as the number of punctured bits grows higher, the proposed layering has an increasing effect on the speed of their recovery. At the code rate 0.9 the E_b/N_0 gain over layered BP decoding with random layering and conventional BP decoding is around 0.7 dB and 0.8 dB at the BER of 10^{-4}, respectively. The proposed layering outperforms random layering at all considered code rates, including the unpunctured mother code. As a result, the check node layering can stay unchanged over all code rates, which further decreases implementation complexity.

Also of interest is the performance gain at different maximum numbers of iterations. Figure 17 shows the required E_b/N_0 values for achieving BER of 10^{-4} at the code rate 0.9 for the maximum numbers of iterations ranging from 10 to 20. When the maximum number of iterations is set to 10 the gain of the proposed layering is 1.3 dB and 1.5 dB over layered BP decoding with random layering and conventional BP decoding, respectively.

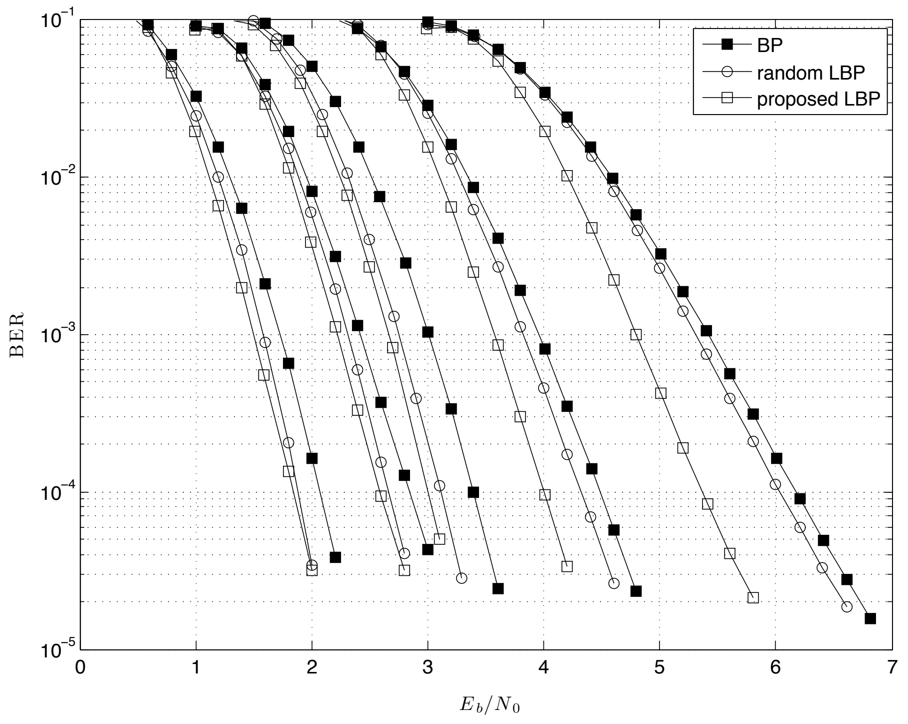

Figure 16: BERs of punctured LDPC codes at code rates 0.5, 0.6, 0.7, 0.8 and 0.9 over an AWGN channel with the maximum number of iterations set to 15. The filled squares, unfilled circles, and unfilled squares represent the BERs of conventional BP decoding, layered BP decoding with random layering, and proposed layering, respectively.

The performance improvements are reduced to about 0.5 dB when the maximum number of iterations is set to 20.

2.6 Concluding Remarks

In this chapter we discussed punctured binary LDPC codes and proposed a grouping algorithm that can be used to design rate compatible punctured LDPC codes at short block lengths. The algorithm is based on the assertion that the performance of punctured LDPC code improves as the level of recoverability of punctured nodes decreases. The concepts of a recovery tree and a recovery error probability are introduced to facilitate the analysis. The proposed puncturing is shown to perform considerably better than random puncturing and the improvements are more pronounced at smaller block lengths. It is worth noting that the ideas behind the grouping algorithm inspired additional research, i.e. [75, 57, 71, 35, 65], which in some cases resulted in additional performance gains.

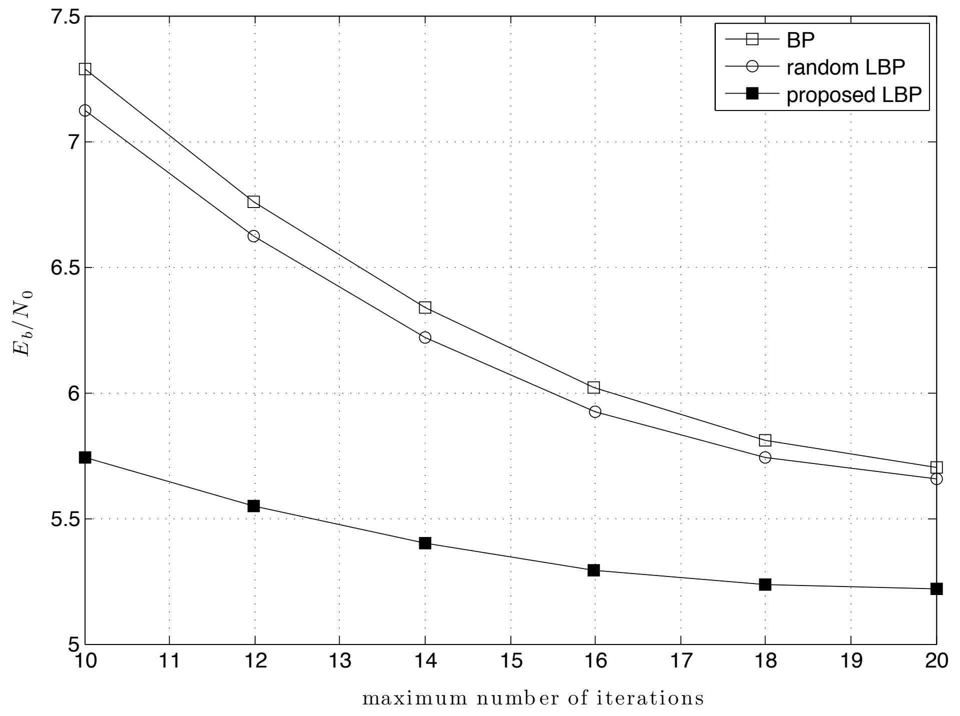

Figure 17: The required E_b/N_0 values for a BER of 10^{-4} at the code rate 0.9 versus the maximum number of iterations, where the LDPC codes are the same as for Figure 16.

We also discussed a solution to slow decoding convergence of punctured LDPC codes. A simple and efficient algorithm is proposed that finds good check node layerings under the layered BP decoding, and yields faster decoding convergence then both layered BP decoding with random layering and conventional BP decoding. This claim is confirmed by simulation results over the AWGN channel. The proposed idea is particularly useful when the maximum number of iterations of LDPC decoders is limited due to practical reasons.

CHAPTER III

PUNCTURED AND SHORTENED NON-BINARY LDPC CODES

3.1 Non-binary LDPC Codes

The main premise in Chapter 2 is that puncturing can be effectively used to obtain a set of well-performing rate-compatible binary LDPC codes. In this chapter we investigate if we can do better. At first glance one might be tempted to think that there is not much need to explore LDPC codes of higher order. The binary LDPC codes are proven to achieve capacity on the BEC channel and were shown to come very close for many others. However, these results only apply at very long block lengths, which are not practical. At practical block lengths, up to a few thousand bits, LDPC codes of higher order, also referred to as *non-binary LDPC codes*, were shown to perform better [12].

The fundamental difference between binary and non-binary LDPC codes is in that non-binary LDPC codes are defined over Galois fields (GF) with more than two elements, while the parity-check matrix of a non-binary LDPC code is still sparse. The following is an example of a parity-check matrix corresponding to an LDPC code over GF(16).

$$\boldsymbol{H} = \begin{bmatrix} 2 & 1 & 0 & 6 & 9 & 3 & 11 & 0 & 0 & 0 \\ 13 & 0 & 0 & 6 & 4 & 0 & 0 & 15 & 8 & 1 \\ 7 & 0 & 3 & 0 & 12 & 0 & 2 & 0 & 5 & 14 \\ 0 & 4 & 9 & 2 & 0 & 10 & 0 & 14 & 9 & 0 \\ 0 & 12 & 1 & 0 & 0 & 10 & 15 & 6 & 0 & 4 \end{bmatrix}$$

Its Tanner graph is constructed in the same manner as in the binary case; except that for each edge we also need to specify the corresponding non-zero element from $\boldsymbol{H}$, called the *edge multiplier*.

While encoding stays basically the same as with binary codes (with the exception that the arithmetic is over a higher order GF), the decoding algorithms are generally considerably more complex. This complexity may have been the principal reason that initially, non-binary

LDPC codes did not gain much acceptance with the research community and industry. Over the last few years, the advent of new decoding algorithms with reduced complexity, e.g. [2, 13, 72] and a better understanding of the design [30, 4, 59] have rekindled interest in non-binary LDPC codes. Also, empirical evidence [30] shows that best performing non-binary LDPC codes are sparser than binary codes and generally have uniform variable node (degree 2) and concentrated check node degree distributions. These properties help to further alleviate the cost in implementation complexity.

A Galois field of the form GF(2^m), where m is a positive integer, is called a *binary extension field*. We can define a bijective mapping ψ from GF(2^m) into $(\mathrm{GF}(2))^m$, which maps elements (or *symbols*) of GF(2^m) into binary m-tuples. Using ψ each element in GF(2^m) can be transmitted over a binary input channel by transmitting its corresponding binary m-tuple. In effect, each non-binary variable node in a Tanner graph gets an extension that accounts for its binary representation, as illustrated in Figure 18. The square node

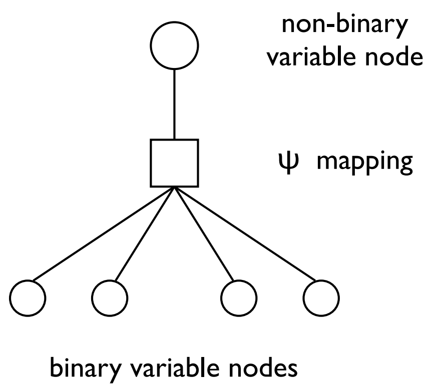

Figure 18: Non-binary variable node with its binary representation.

represents the mappings ψ and ψ^{-1}, ψ for the messages to binary variable nodes and ψ^{-1} for the messages to non-binary variable nodes. With this extension, a Tanner graph of a non-binary LDPC code looks the example in Figure 19.

In this chapter, we analyze non-binary LDPC codes over binary extension fields and their performance over binary input channels. Of particular interest is their ability to adapt the coding rate to different channel conditions with a single encoder/decoder pair

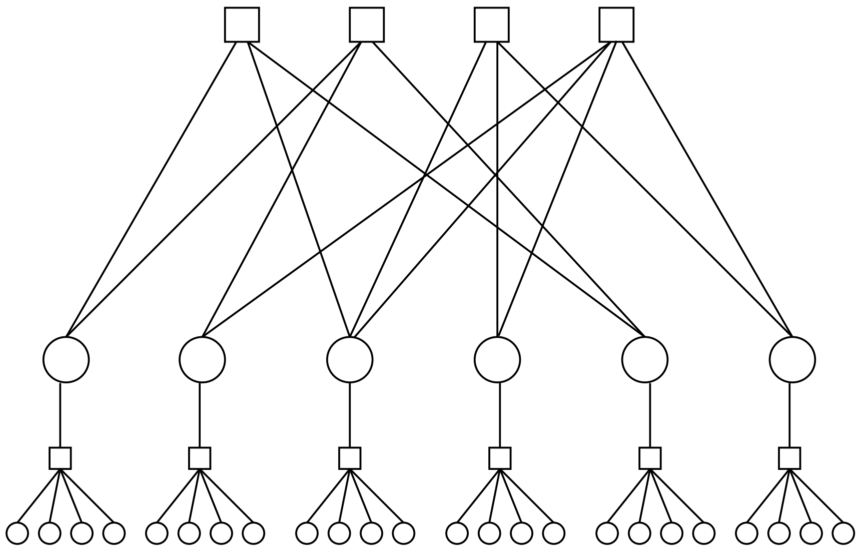

Figure 19: An example of a non-binary Tanner graph.

by using puncturing. In addition, we will examine a method called *shortening*, used to obtain lower-rate codes from a mother code. To shorten a mother code means to choose a subset of message-carrying variable nodes and to set them to a predetermined value, for instance 0. Shortened variable nodes are known in advance to both the transmitter and the receiver; hence, there is no need to transmit them. Notice that if we shorten K_s out of K message-carrying variable nodes in an LDPC code with N variables nodes, the code rate becomes

$$R_s = \frac{K - K_s}{N - K_s}, \tag{38}$$

which is smaller than the rate $R = K/N$ of the unpunctured mother code.

It is interesting to note that a fair amount of empirical results indicate that non-binary LDPC codes over GF(q), particularly when $q \geq 32$, perform best when their variable node degree distribution is regular with degree 2 [30]. With this in mind, we restrict our analysis to regular non-binary LDPC codes with a variable node degree 2 throughout this chapter.

3.2 The Puncturing Problem

The puncturing problem in the previous chapter was to determine variable nodes to be punctured for best performance. The grouping algorithm can be applied for puncturing non-binary LDPC codes as well, if the incidence matrix of the non-binary Tanner graph is used as the parity check matrix. The algorithm would determine the variable nodes to be punctured and we would puncture all bits in their binary representations.

However, the puncturing problem with non-binary codes used over binary-input channels gains another degree of freedom. In addition to choosing variable nodes to puncture, it must be decided which bits in their binary representations should be punctured. For example, in a non-binary LDPC code over GF(16) assume we have to puncture four bits (see Figure 20). One approach is to choose one variable node and puncture all four bits in its binary representation. Alternatively, we can choose two variable nodes and puncture only two bits per variable node. Or choose three variable nodes and puncture two bits on one variable node, and two bits on the remaining two. In all these cases four bits would be punctured.

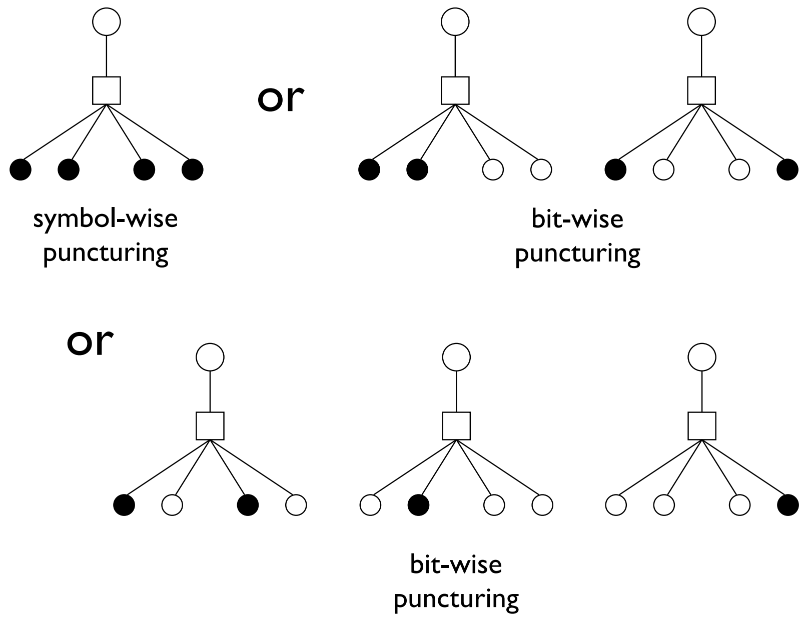

Figure 20: Symbol-wise and bit-wise puncturing.

We define the *puncturing pattern* to be the set of all bits and variable nodes that are set to be punctured. If at least one bit of a variable node is punctured, we say that that variable node is in the puncturing pattern. A variable node is said to be punctured *entirely*, if all bits in its binary representation are punctured, or *partially*, if only some are punctured. If all variable nodes in a puncturing pattern are punctured entirely, we say the code is punctured *symbol-wise*, otherwise it is punctured *bit-wise*. Further, we define the notion of *puncturing depth*, which is used for each variable node individually and indicates how many bits in its binary representation were punctured. For instance, if 3 bits of a variable node were punctured, we say that the symbol's puncturing depth is 3. Analogous definitions can be used for shortening as well.

3.3 Puncturing and Shortening Over the BEC

The most powerful method for design and performance analysis of LDPC codes is *density evolution* [60], which tracks the average probability density function of messages during the decoding process and is computationally very intense. In [20] it was used to design asymptotically optimal puncturing distributions for binary LDPC codes. Unfortunately, density evolution proves to be computationally too intense for non-binary LDPC codes over most channels. An exception is the binary erasure channel (BEC), where the density evolution can be reduced to tracking of a single parameter [59], and thus becomes computationally feasible. We extend the results in [59] to account for puncturing and shortening and introduce a framework that enables optimization of puncturing/shortening distributions for a wide range of rates via differential evolution, an non-linear optimization technique introduced in [67]. We show that symbol-wise puncturing is generally results in poor performance. Instead, while some variable nodes may be punctured entirely, most of them should be punctured only partially to achieve best performance.

It is worth reminding that, asymptotically, non-binary LDPC codes do not outperform binary LDPC codes over the BEC, as binary LDPC codes achieve capacity both when unpunctured [49] and punctured [58]. Nevertheless, for binary LDPC codes many results obtained over the BEC channel provided useful insight into their behavior over general memoryless symmetric channels [63]. We anticipate that similar would apply to non-binary LDPC codes.

In order to describe puncturing distributions we define $m+1$-tuples

$$\boldsymbol{\pi_i} = [\pi_{i,0} \quad \pi_{i,1} \quad \cdots \quad \pi_{i,m}] \tag{39}$$

for each distinct degree i in a variable node degree distribution, where $\pi_{i,j}$ denotes the fraction of variable nodes of degree i and puncturing depth j. For a given set of $\boldsymbol{\pi_i}$'s and for a mother code of rate R, the overall fraction of punctured bits $p^{(0)}$ is given by

$$p^{(0)} = \frac{1}{m}\sum_{i=2}^{d_v} \Lambda_i \sum_{j=1}^{m} j\,\pi_{i,j}. \tag{40}$$

The new increased rate after puncturing is then

$$R_p = \frac{R}{1-p^{(0)}}. \tag{41}$$

Similarly, to describe shortening distributions we define $m+1$-tuples

$$\boldsymbol{\sigma_i} = [\sigma_{i,0} \quad \sigma_{i,1} \quad \cdots \quad \sigma_{i,m}] \tag{42}$$

for each distinct degree i in a variable node degree distribution, where $\sigma_{i,j}$ denotes the fraction of variable nodes of degree i and shortening depth j. For a given set of $\boldsymbol{\sigma_i}$'s and for a mother code of rate R the overall fraction of shortened bits is given by

$$s^{(0)} = \frac{1}{m}\sum_{i=2}^{d_v} \Lambda_i \sum_{j=0}^{m} j\,\sigma_{i,j}. \tag{43}$$

The new decreased rate after shortening is then

$$R_s = \frac{R-s^{(0)}}{1-s^{(0)}}. \tag{44}$$

The Gaussian binomial coefficients, used in the next subsection, are defined as follows

$$\begin{bmatrix} m \\ k \end{bmatrix} = \begin{cases} 1, & \text{if } k=0 \text{ or } k=m, \\ \prod_{l=0}^{k-1} \frac{2^m-2^l}{2^k-2^l}, & \text{otherwise.} \end{cases} \tag{45}$$

This Gaussian binomial coefficient is the number of subspaces of dimension k of a vector space over the field GF(2^m).

3.3.1 Density Evolution Over the BEC

For non-binary LDPC codes over the field GF(2^m) a message sent over an edge in the Tanner graph is a vector with 2^m components, where the i-th component is the probability that the variable node connected to the edge is the i-th element of the non-binary field. As was observed in [59], density evolution for these codes is greatly simplified over the BEC channel.

Suppose an m-bit symbol is transmitted over the BEC, where k bits are erased while the remaining $m-k$ bits are received error-free. From receiver's point of view any combination of bits at k erased positions is possible; therefore, the decoder sees each of the 2^k possible combinations as equally probable. We say that a message has dimension k if it has 2^k non-zero components. All the remaining $2^m - 2^k$ components then have probability 0. It is shown in [59, Lemma 4.2] that in a belief propagation decoder over BEC all non-zero components of a message have equal probabilities through all iterations; therefore, it suffices to track the dimension of a message through iterations and density evolution can be expressed by means of a $(m+1)$-dimensional recursion. The remainder of this subsection is a summary of results in [59].

Let $P_v^{(l)}(k,r)$ be the probability that, in l-th iteration, a randomly chosen message emanating from a variable node of degree r after permutation due to an edge multiplier has dimension k. Further, let $P_c^{(l)}(k,r)$ be the probability that, in l-th iteration, a randomly chosen message emanating from a check node of degree r, after permutation due to an edge multiplier, has dimension k. Then there are the following recursive relationships between probabilities on the check node side

$$P_c^{(l)}(k,3) = \sum_{i=0}^{k} P_v^{(l)}(i) \sum_{j=k-i}^{k} A_c(m,i,j,k,l) \tag{46}$$

$$P_c^{(l)}(k,r) = \sum_{i=0}^{k} P_c^{(l)}(i,r-1) \sum_{j=k-i}^{k} A_c(m,i,j,k,l) \tag{47}$$

with

$$A_c(m,i,j,k,l) = \frac{\begin{bmatrix} m-i \\ m-k \end{bmatrix} \begin{bmatrix} i \\ k-j \end{bmatrix}}{\begin{bmatrix} m \\ m-j \end{bmatrix}} 2^{(k-i)(k-j)} P_v^{(l)}(j) \tag{48}$$

and $P_v^{(l)}(j)$ is the average over the variable node degree distribution $\lambda(x)$

$$P_v^{(l)}(j) = \sum_{i=2}^{d_v} \lambda_i P_v^{(l)}(j,i) \tag{49}$$

The probability $P_c^{(l)}(k,r)$ is initially evaluated for two incoming variable node messages in Eq. (46) and thereafter, if a check node's degree is higher than 3, $P_c^{(l)}(k,r)$ for each additional incoming variable node message is calculated recursively according to Eq. (47). Suppose we want to find out the a probability that the dimension of a sum of two messages is k; if the first message has dimension i, $i \leq k$, the dimension j of the second message must satisfy $k-i \leq j \leq k$. The term $A_c(m,i,j,k,l)$ denotes the probability that the 2^j non-zero components of the second message are arranged such that the sum of two messages has dimension k.

On the variable node side the equations for the probabilities in $(l+1)$-th iteration are

$$\begin{aligned} P_v^{(l+1)}(k,2) &= \sum_{i=k}^{m} \binom{m}{i} \epsilon^i (1-\epsilon)^{m-i} \sum_{j=k}^{m-i+k} A_v(m,i,j,k,l) & (50) \\ P_v^{(l+1)}(k,r) &= \sum_{i=k}^{m} P_v^{(l+1)}(i,r-1) \sum_{j=k}^{m-i+k} A_v(m,i,j,k,l) & (51) \end{aligned}$$

where

$$A_v(m,i,j,k,l) = \frac{\begin{bmatrix} i \\ k \end{bmatrix} \begin{bmatrix} m-i \\ j-k \end{bmatrix}}{\begin{bmatrix} m \\ j \end{bmatrix}} 2^{(i-k)(j-k)} P_c^{(l)}(j) \tag{52}$$

and $P_c^{(l)}(j)$ is the average over the check node degree distribution $\rho(x)$

$$P_c^{(l)}(j) = \sum_{i=2}^{d_c} \rho_i P_c^{(l)}(j,i)$$

The probability that the initial message from a variable node has dimension k equals

$$P_v^{(0)}(k) = \binom{m}{k} \epsilon^k (1-\epsilon)^{m-k}.$$

For a given degree distribution pair $(\lambda(x), \rho(x))$, the threshold ϵ_{th} is the maximum erasure probability ϵ on the BEC channel, for which

$$\begin{aligned} \lim_{l \mapsto \infty} P_v^{(l)}(0) &= 1 \\ \lim_{l \mapsto \infty} P_v^{(l)}(k) &= 0 \qquad \text{for all } 1 \leq k \leq m. \end{aligned}$$

3.4 Optimization of Puncturing and Shortening Distributions

We investigate puncturing and shortening of non-binary LDPC codes in the asymptotic sense (for very long block lengths) over the BEC channel. As we will use density evolution to evaluate performance, the equations of Section 3.3.1 have to be modified to account for puncturing and shortening.

Let us start with puncturing. Suppose an m-bit symbol has puncturing depth p. The dimension of the corresponding message will initially be p or higher if some of the $m-p$ transmitted bits are erased. With systematic puncturing, the probability that an initial message from a variable node is of a certain dimension depends on that nodes' degree. Consequently, the recursion in Eqs. (50) and (51) must be performed for each variable node degree separately. For a group of nodes of degree d this recursion becomes

$$P_{v,d}^{(l+1)}(k,2) = \sum_{i=k}^{m} P_{v,d}^{(0)}(i) \sum_{j=k}^{m-i+k} A_v(m,i,j,k,l) \tag{53}$$

$$P_{v,d}^{(l+1)}(k,r) = \sum_{i=k}^{m} P_{v,d}^{(l+1)}(i,r-1) \sum_{j=k}^{m-i+k} A_v(m,i,j,k,l) \tag{54}$$

where $P_{v,d}^{(0)}(i)$ is the probability that the initial message from a randomly chosen variable node of degree d has dimension i. We have

$$P_{v,d}^{(0)}(k) = \sum_{i=0}^{k} \binom{m-i}{k-i} \pi_{d,i}\epsilon^{k-i}(1-\epsilon)^{m-k}, \tag{55}$$

where $\pi_{d,0} = 1 - \sum_{i=1}^{m} \pi_{d,i}$ denotes the fraction of unpunctured variable nodes of degree d. The initial message will have dimension i if p bits are punctured and $i-p$ bits are erased on the channel for any $0 \leq p \leq i$. Puncturing also changes Eq. (49) to

$$P_v^{(l)}(j) = \sum_{i=2}^{d_v} \lambda_i P_{v,i}^{(l)}(j,i) \tag{56}$$

Next, we investigate shortening. Suppose an m-bit symbol has shortening depth s; then, at initialization in the decoder s bits are guaranteed to be known and the dimension of the initial message can be at most $m-s$. Similarly as above, the recursion in Eqs. (50) and (51) must be performed for each variable node degree separately, where for shortening we have

$$P_{v,d}^{(0)}(k) = \sum_{i=0}^{m-k} \binom{m-i}{k} \sigma_{d,i} \, \epsilon^k (1-\epsilon)^{m-i-k}, \tag{57}$$

and $\sigma_{d,0} = 1 - \sum_{i=1}^{m} \sigma_{i,p}$ denotes the fraction of unshortened variable nodes of degree d. The probability $P_v^{(l)}(j)$ is obtained according to Eq. (56).

The optimization of the puncturing distribution is carried out as follows. For a given regular mother code with degree distribution pair $\lambda(x) = x, \rho(x) = x^{d_c-1}$ of rate R and the desired fraction of punctured bits $p^{(0)}$, maximize the threshold

$$\max_{\boldsymbol{\pi_2}} \epsilon_{\text{th}},$$

subject to constraints

$$\frac{\sum_{i=1}^{m} i\, \pi_{2,i}}{m} = p^{(0)} \text{ and } \sum_{i=1}^{m} \pi_{2,i} \le 1.$$

Similarly, for a given fraction of shortened bits $s^{(0)}$, the optimization problem is to maximize the threshold

$$\max_{\boldsymbol{\sigma_2}} \epsilon_{\text{th}},$$

subject to constraints

$$\frac{\sum_{i=1}^{m} i\, \sigma_{2,i}}{m} = s^{(0)} \text{ and } \sum_{i=1}^{m} \sigma_{2,i} \le 1.$$

3.4.1 Simulation Results

We proceed to investigate the performance of rate-adaptive non-binary LDPC codes designed with the proposed framework and compare it with some alternate design approaches. We choose a regular LDPC code over GF(2^6) with degree distribution pair $\lambda(x) = x, \rho(x) = x^3$ and optimize its puncturing and shortening distributions over a range of rates from 0.091 to 0.909. Recall that non-binary LDPC codes are believed to perform best when all their variable nodes have degree 2; hence, the choice for the mother code.

We vary fractions of punctured and shortened bits $p^{(0)}$ and $s^{(0)}$ from 0.05 to 0.45 in steps of 0.05, and optimize distributions for each fraction with differential evolution [67]. The results, together with all thresholds, are given in Tables 4 and 5.

For comparison, we design rate-adaptable codes using two alternate approaches: one where we puncture/shorten the mother code symbol-wise (6 bits per symbol), and the

Table 4: Optimized puncturing distributions obtained via differential evolution. Mother code is a regular (2,4), rate 0.5, LDPC code over GF(2^6).

$p^{(0)}$	0	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45
rate	0.5	0.5263	0.5556	0.5882	0.6250	0.6667	0.7143	0.7692	0.8333	0.9091
$\pi_{2,1}$	0.00000	0.00722	0.22891	0.51850	0.76528	0.88677	0.58875	0.31502	0.07773	0.00041
$\pi_{2,2}$	0.00000	0.00068	0.00064	0.00798	0.00346	0.01412	0.31195	0.57751	0.80031	0.74116
$\pi_{2,3}$	0.00000	0.00265	0.00074	0.00009	0.00039	0.00030	0.00070	0.00394	0.00115	0.10915
$\pi_{2,4}$	0.00000	0.00047	0.00010	0.00043	0.00148	0.00111	0.00081	0.00033	0.00011	0.00267
$\pi_{2,5}$	0.00000	0.00080	0.00028	0.00082	0.00034	0.00009	0.00045	0.00011	0.00009	0.00005
$\pi_{2,6}$	0.00000	0.04626	0.06097	0.05991	0.06984	0.09653	0.09662	0.10271	0.11955	0.14648
% punct.symb.	0.00	5.81	29.16	58.77	84.08	99.89	99.93	99.96	99.89	99.99
ϵ_{th}	0.4746	0.4489	0.4185	0.3847	0.3474	0.3054	0.2561	0.2002	0.1366	0.0607

Table 5: Optimized shortening distributions obtained via differential evolution. Mother code is a regular (2,4), rate 0.5, LDPC code over GF(2^6).

$s^{(0)}$	0	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45
rate	0.5	0.4737	0.4444	0.4118	0.3750	0.3333	0.2857	0.2308	0.1667	0.0909
$\sigma_{2,1}$	0.00000	0.01281	0.01029	0.00599	0.00600	0.00189	0.00284	0.00158	0.00017	0.00030
$\sigma_{2,2}$	0.00000	0.08612	0.10069	0.32999	0.24670	0.16036	0.04052	0.00471	0.00015	0.00036
$\sigma_{2,3}$	0.00000	0.03298	0.12435	0.06993	0.22820	0.39056	0.56579	0.69001	0.65232	0.51284
$\sigma_{2,4}$	0.00000	0.00315	0.00247	0.00428	0.00385	0.00061	0.00370	0.00384	0.10973	0.28933
$\sigma_{2,5}$	0.00000	0.00051	0.00026	0.00102	0.00000	0.00064	0.00370	0.00053	0.00023	0.00058
$\sigma_{2,6}$	0.00000	0.00015	0.00069	0.00034	0.00010	0.00000	0.00042	0.00016	0.00041	0.00004
% short.symb.	0.00	13.57	23.87	41.15	48.49	55.41	61.36	70.08	76.30	80.35
ϵ_{th}	0.4746	0.4998	0.5276	0.5590	0.5943	0.6345	0.6808	0.7349	0.7977	0.8727

other where we puncture/shorten bit-wise uniformly, so that each punctured/shortened symbol has puncturing/shortening depth 3. The gap to capacity of these codes is shown in Figure 21.

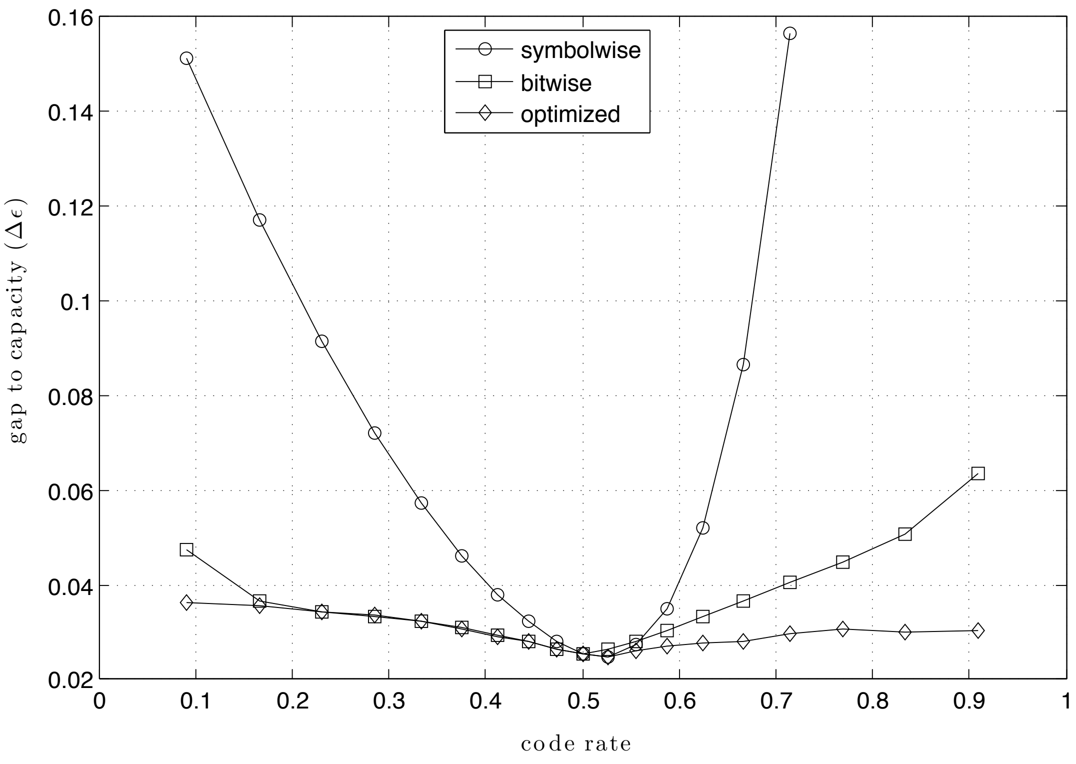

Figure 21: Performance comparison between the proposed optimized codes (DiffEv), symbol-wise shortening and puncturing (6bps), and uniform bit-wise shortening and puncturing (3bps).

Symbol-wise puncturing and shortening result in poor performance. The overall fraction of punctured symbols is minimized, but the high uncertainty in the decoder has unfavorable consequences, especially at high puncturing fractions. The highest achievable rate with a positive threshold is barely over 0.7. On the other hand, spreading punctured and shortened bits improves performance. Gains are present at both ends of the code rate range. Nevertheless, the best performance is achieved with optimized puncturing and shortening distributions, where a small gap to capacity is maintained over the entire range of considered code rates.

There is always a notable fraction of variable nodes that are punctured entirely, while the remaining bits are spread over the remaining variable nodes so that their depth is small.

For $p^{(0)}$ of 0.25 and higher, the puncturing pattern is spread over all variable nodes (see row "% punct. symbs" in Table 4). The nonuniformity of the puncturing pattern can be seen as an irregular structure imposed on a regular LDPC code that results in significant improvements in performance.

To verify the results in Tables 4 and 5, we designed a regular (2,4) LDPC code over GF(2^6) with a block length of 20000 symbols (or 120000 bits), punctured/shortened it

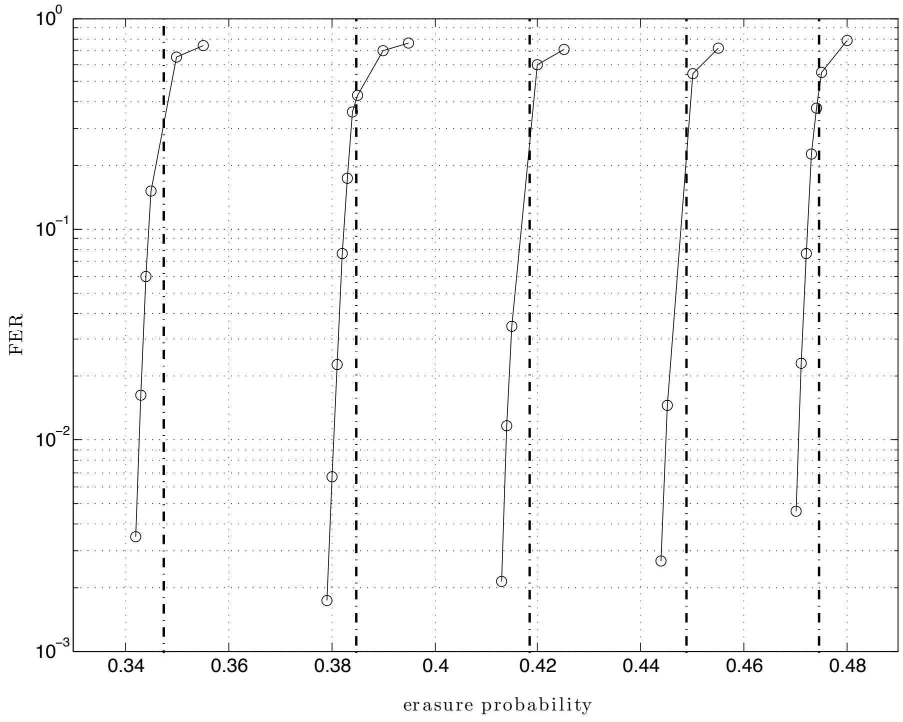

Figure 22: Frame-error-rate (FER) performance of punctured LDPC codes; rates from right to left: 0.5 (unpunctured), 0.52, 0.56, 0.58, and 0.62. The dashed lines represent the corresponding thresholds.

according to optimized puncturing distributions, and simulated the resulting performance over the BEC channel[1]. The results along with the calculated thresholds are presented in Figures 22—25. The simulated performance matches well with analytical results at all considered code rates.

Performance variation depending on the field size is shown in Figure 26, where the Galois fields GF(2^4), GF(2^6) and GF(2^{10}) are considered. Mother code performance improves with

[1]Maximum number of iterations was 130.

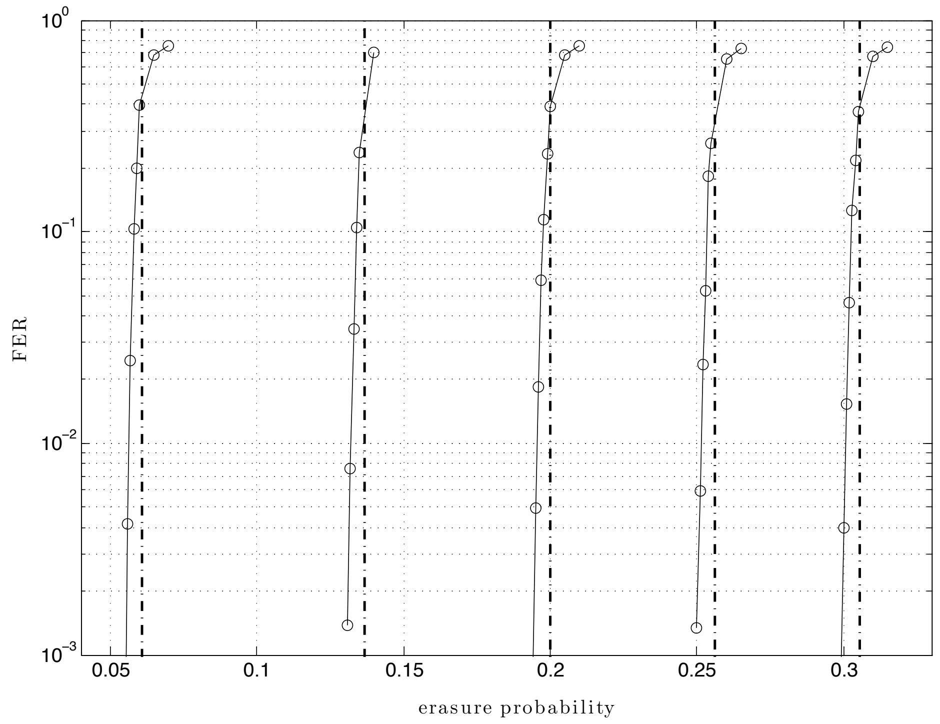

Figure 23: FER performance of punctured LDPC codes; rates from right to left: 0.66, 0.71, 0.76, 0.83, and 0.91. The dashed lines represent the corresponding thresholds.

size of the field until GF(2^6) and then starts declining again. In fact, the mother code has a lower threshold over GF(2^{10}) than over GF(2^4). While performance of unpunctured codes does not always improve with the increasing field size [59], it appears that that the code's ability to recover from optimized puncturing improves with the field size. When puncturing is used the gap to capacity very quickly drops to values around 0.02 over the field GF(2^{10}).

3.4.2 Highest Achievable Rate

The cut-off rate, which is the highest achievable rate that can be achieved by means of puncturing, was analyzed for punctured binary LDPC codes in [58]; it was observed that performance degrades considerably when the rate approaches the cut-off rate. In Figure 27 we investigate the cut-off rate for punctured non-binary LDPC codes over the BEC, where we find that it can be improved significantly if the code is punctured bit-wise instead of symbol-wise. For symbol-wise puncturing the cut-off rate does not depend on the field dimension m. On the other hand, for bit-wise puncturing the cut-off rate rises steadily with

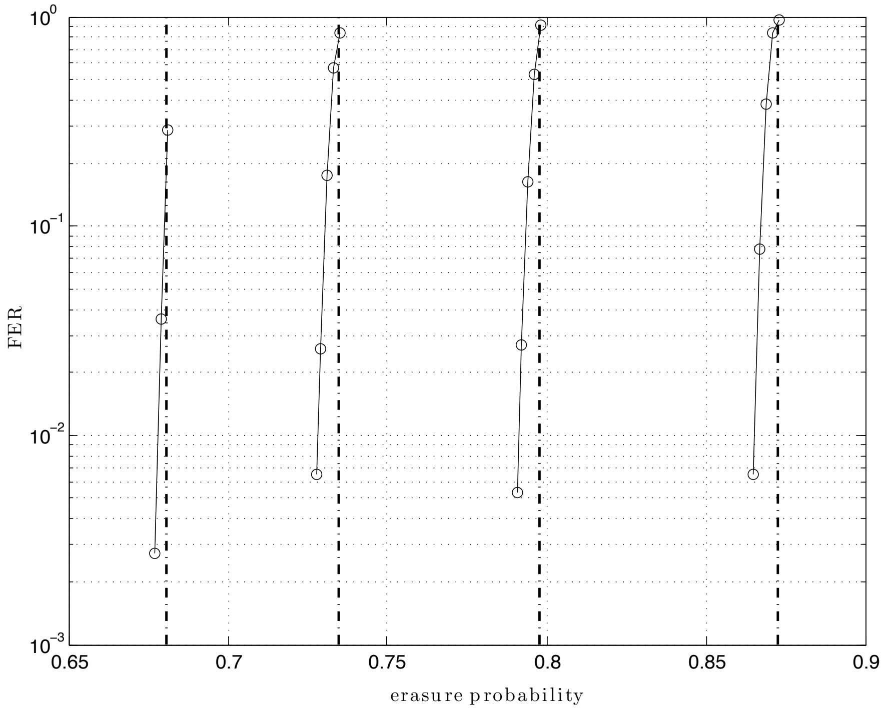

Figure 24: FER performance of shortened LDPC codes; rates from right to left: 0.09, 0.17, 0.23, and 0.29. The dashed lines represent the corresponding thresholds.

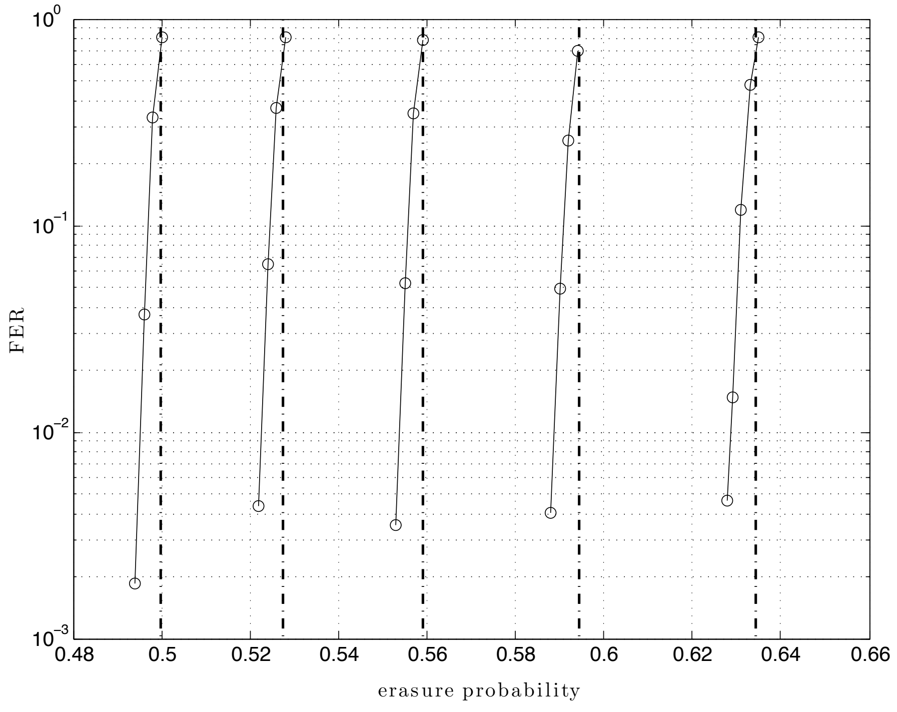

Figure 25: FER performance of shortened LDPC codes; rates from right to left: 0.33, 0.38, 0.41, 0.44, and 0.47. The dashed lines represent the corresponding thresholds.

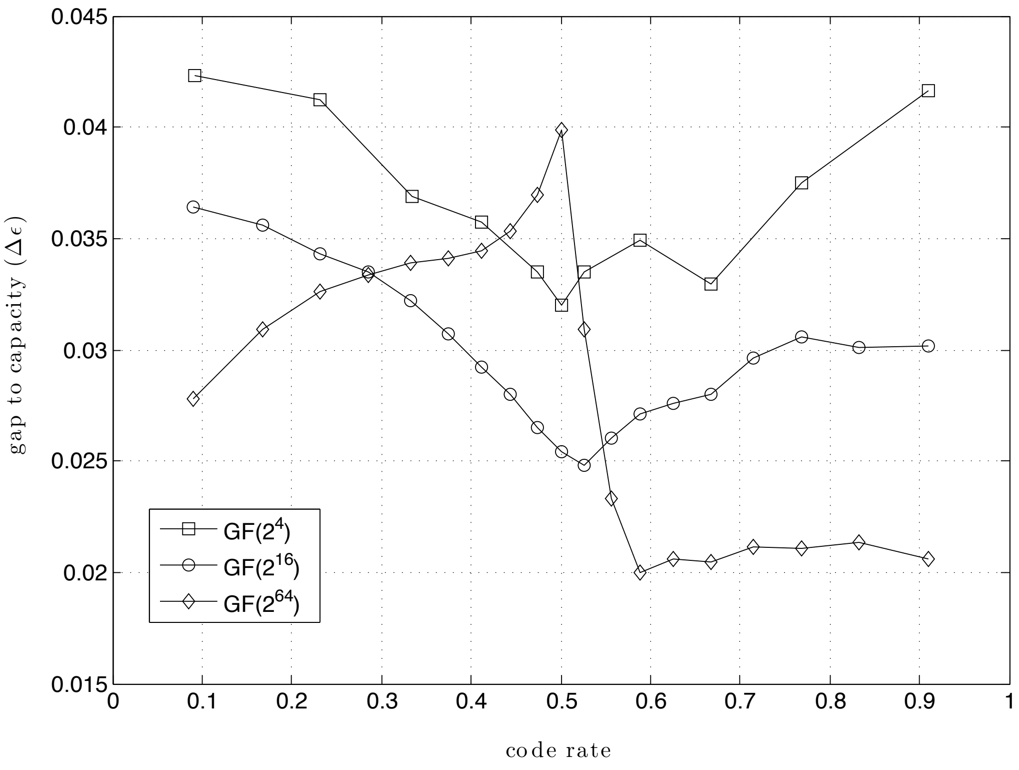

Figure 26: Performance of regular rate-adaptive non-binary LDPC codes over GF(2^4), GF(2^6) and GF(2^{10}).

growing m. Most likely this behavior can be attributed to higher flexibility of available puncturing depths that can be imposed upon individual symbols. Once more, the irregular structure of the punctured code imposed by bit-wise puncturing proves to be beneficial.

3.5 Puncturing and Shortening at Finite (Short) Block Lengths

Non-binary LDPC codes with very long block lengths are not well suited for most practical applications due to intolerable cost in implementation complexity and delay. Moreover, the benefit of using non-binary over binary LDPC codes is most pronounced at short block lengths. Experience from binary LDPC codes shows that using asymptotically optimized punctured distributions for codes with short block lengths does not result in good performance. It is much better to design specialized puncturing algorithms that are applied to specific instances (not ensembles) of LDPC codes in order to determine the best puncturing locations.

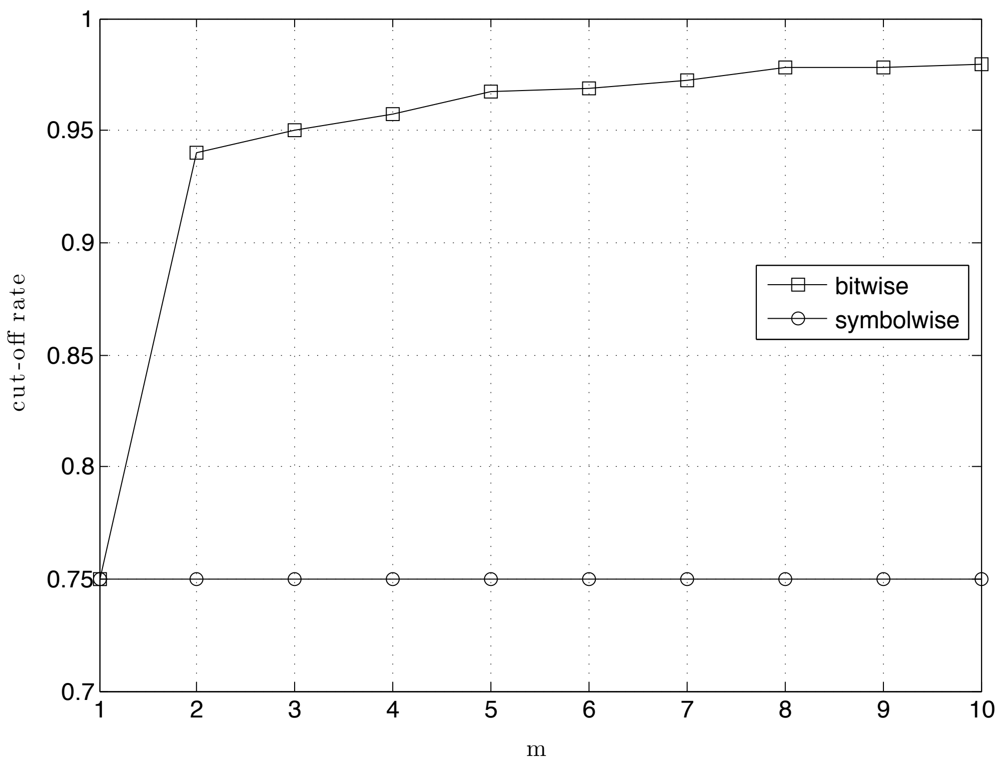

Figure 27: Cut-off rate vs. parameter m.

Similar conclusions hold for non-binary LDPC codes as well. In this section we propose algorithms for puncturing and shortening of non-binary LDPC codes with short block lengths and show that they consistently outperform puncturing/shortening according to optimized puncturing distributions and that rate-compatible non-binary LDPC codes exhibit a performance superior to that of rate-compatible binary LDPC codes. The evaluations will be performed on two common channel models: BEC and AWGN.

Consider a non-binary LDPC code over GF(2^m) over a binary input channel. The messages in a message-passing decoder for non-binary LDPC codes are vectors with $q = 2^m$ components that characterize the probability for each of q symbols. Since the sum of all components equals 1, $q - 1$ components are sufficient to characterize the probability distribution. Following [72], for a random variable v over GF(2^m) define its *log-likelihood ratio vector* (LLRV) as

$$\boldsymbol{L}(v) = [L(v = \alpha_1), \ldots, L(v = \alpha_{q-1})],$$

where $q = 2^m$ and

$$L(v = \alpha_i) = \ln \frac{\Pr[v = \alpha_i]}{\Pr[v = 0]}.$$

We will use a message-passing decoder algorithm in the log-likelihood domain introduced by Wymeersch *et al.* [72], where messages exchanged between variable and check nodes are LLRVs.

Assume that a non-binary LDPC code is punctured. In Chapter 2, we referred to a punctured variable node as *recovered* after it received the first non-zero message from one of its neighboring check nodes. With non-binary codes, where messages are vectors (not scalars), we need to be more specific. A punctured non-binary variable node will be called k-SR, if the first LLRV without zero components from one of its neighboring check nodes arrives in the k-th iteration. The notions of *level of recoverability* and *maximum level of recoverability* now apply analogously to Chapter 2.

Consider a variable node with value 0 in GF(2^m) that is represented by an all-zero binary sequence of length m. This sequence is punctured before it is transmitted over a noiseless channel; that is, all transmitted bits are observed perfectly at the receiver. If p out of m bits are punctured, the initial LLRV at the receiver for this variable node will have $2^p - 1$ components with value 0, one for each symbol whose binary representation has 1's at the punctured locations. In effect, the number of zero components in the initial LLRV grows exponentially with the number of punctured bits per symbol (see Table 6).

Table 6: Number of zero components in the initial LLRV for a punctured symbol in GF(2^6)

punctured bits per symbol	1	2	3	4	5	6
zero components	1	3	7	15	31	63

For instance, consider an LDPC code over GF(2^6). Puncturing 5 bits in a symbol rather than all 6 reduces the number of zero components in the initial LLRV from 63 to 31, while reducing the number of punctured bits per symbol from 2 to 1 only slightly reduces the number of zero components. Decreasing the number of punctured bits per variable node decreases the initial uncertainty the decoder; yet, it also increases the number of partially punctured variable nodes. There seems to be a tradeoff between the overall number of punctured variable nodes (entirely or partially) and the initial uncertainty of the punctured variable nodes in the decoder.

3.5.1 Puncturing Algorithm

Our observations in numerous experiments with short block length non-binary codes over BEC and AWGN channels indicate that puncturing many bits in symbols with high levels of recoverability has increasingly detrimental effects on performance. The observations also suggest that at intermediate rates, where we have the flexibility, bit-wise puncturing should be employed rather than symbol-wise puncturing.

Therefore we propose a two-step puncturing algorithm: in the first step, symbols subject to puncturing are selected using the grouping algorithm. Subsequently, the punctured bits are spread over the symbols that were chosen in the first step. A more detailed description is as follows:

Step 1 [**Symbol Grouping**] It is important that punctured symbols are recovered as quickly as possible. Toward this end the grouping algorithm from Section 2.3 is applied on the incidence matrix of a given non-binary Tanner graph to choose variable nodes to be punctured. The algorithm starts by searching for 1-SR nodes and attempts to maximize their number; when no more 1-SR nodes can be found, it proceeds with 2-SR nodes, and so on, until every variable node is set to be either punctured or unpunctured.

Step 2 [**Bitwise Spreading at Intermediate Rates**] At the highest code rate there is not much flexibility as all variable nodes chosen in Step 1 must be punctured entirely. At intermediate rates puncturing is spread bit-wise over symbols chosen in Step 1. The symbols with low levels of recoverability are more resilient to puncturing; therefore, they are assigned higher puncturing depths then those with higher levels of recoverability. The puncturing depth should be kept uniform over all symbols with the same level of recoverability. If the intermediate puncturing patters are required to be rate-compatible, some additional care is needed at intermediate rates, but generally this is done fairly easily.

3.5.2 Shortening Algorithm

We proceed to show how the rate can be lowered by shortening information bits. Let K be the number of variable nodes carrying information symbols, N the number of all variable nodes, K_s the number of shortened variable nodes, and m the number of bits per symbol. Then the rate after shortening R_s is given by

$$R_s = \frac{m(K - K_s)}{N - K_s}. \tag{58}$$

Much like puncturing, shortening non-binary LDPC codes gains a degree of freedom: shortening depth. We will show that bit-wise shortening yields better performance than symbol-wise shortening. Shortened bits should be spread over all variable nodes carrying information bits such that the number of shortened bits per information variable node is as uniform as possible. Assuming that mK_s bits need to be shortened, the shortening algorithm is as follows:

Step 1 [**Spread**] $a = \lfloor mK_s/K \rfloor$, $i_1 = mK_s - K \cdot a$, $i_2 = K - i_1$;

Step 2 [**Randomize**] Randomly choose i_1 information variable nodes and shorten $a + 1$ bits on each of them;

Step 3 [**Shorten**] Shorten a bits on the remaining i_2 information variable nodes.

This strategy closely follows asymptotically optimal shortening distributions from Table 5, except that here the shortened bits are spread only over K information symbols (which simplifies encoding). The asymptotically optimal shortening distributions sometimes demand that more than K symbols be shortened, which requires additional work to ensure that encoding remains feasible.

3.5.3 Simulation Results

To verify the effectiveness of the proposed algorithms for puncturing and shortening over short block lengths, we design a system that supports code rates ranging from 0.23 to 0.91. For a representative of non-binary LDPC codes we choose a regular (2, 4) mother code over

GF(2^6) with a block length of 142. Since each symbol in GF(2^6) is represented by 6 bits, transmitting one unpunctured codeword over a binary input channel requires 852 channel uses. As its binary competitor, we choose the irregular binary LDPC code from [70] with a block length of 852 and the degree distribution pair:

$$\begin{aligned}\lambda(x) &= 0.25105x + 0.30938x^2 + 0.00104x^3 + 0.43853x^9,\\ \rho(x) &= 0.63676x^6 + 0.36324x^7.\end{aligned}$$

Parity check matrices for binary and non-binary codes were generated using the Progressive Edge Growth (PEG) algorithm [30]. The edge multipliers for the non-binary code were chosen randomly from all non-zero GF(64) elements for each edge and the number of maximum iterations in all simulations was set to 50. The proposed algorithms for puncturing and shortening are evaluated on BEC and AWGN channels and compared against two puncturing/shortening approaches:

- *symbol-wise:* puncturing and shortening are performed symbol-wise. The symbols subject to shortening are chosen randomly among the independent (information) symbols, while punctured symbols are chosen according to the grouping algorithm (applied to the incidence matrix of the non-binary code);

- *asymptotically optimized:* puncturing/shortening patterns are obtained using the asymptotically optimized distributions from Section 3.4.1.

We address puncturing first. We created a puncturing pattern that achieves the code rate 0.91 for both codes using the algorithm from [22]. The group distribution is shown in Table 7.

In order to evaluate our puncturing approach we design four puncturing patterns at the code rate 0.83 and evaluate their performance over the AWGN channel. In the first puncturing pattern, punctured bits are chosen at random; in the second, puncturing is performed symbol-wise according to [22]; in the third, the punctured bits are selected randomly among variable nodes selected in Step 1; and in the fourth, puncturing is performed using the proposed algorithm in Section 3.5.1.

Table 7: The puncturing pattern achieving code rate 0.91 by levels of recoverability of variable nodes for the considered binary and GF(2^6) LDPC code.

Level of Recoverability	Binary	GF(2^6)
unpunctured	468	78
1	200	48
2	66	12
3	55	4
4	36	/
5	23	/
6	4	/
all	852	142

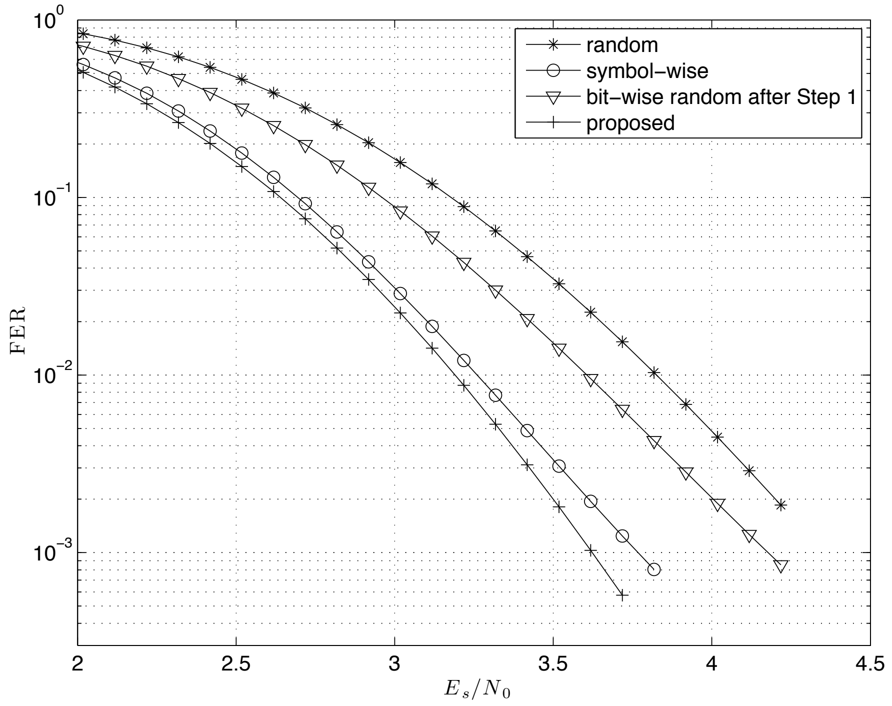

Figure 28: Performance comparison between four differently chosen puncturing patterns at rate 0.83.

Figure 28 shows the results. Observe that puncturing bits randomly is not a good approach as it yields poor performance and at high rates some puncturing patterns may produce unrecoverable symbols. This motivates the decision that spreading is to be performed only over symbols that are guaranteed to be recovered quickly. Symbol-wise puncturing according to [22] ensures that all punctured bits are recovered quickly, which results in a significantly improved performance. Random bit-wise puncturing after Step 1 performs worse than symbol-wise puncturing, while the best performance is achieved by the proposed puncturing algorithm and thus, our approach is validated.

Next we evaluate performance of the proposed puncturing algorithm at intermediate code rates 0.62, 0.71, and 0.82. At the highest considered rate of 0.91 all symbols must be punctured entirely; therefore, both the proposed puncturing algorithm and the symbol-wise puncturing yield the same puncturing pattern. The puncturing patterns obtained by the proposed puncturing algorithm for each rate are given in Table 8.

Table 8: The breakdown of puncturing patterns obtained by the proposed algorithm for the non-binary LDPC code over GF(2^6). Except in the bottom row, the listed numbers indicate the number of variable nodes.

Level	# of punct. bits	0.62	0.71	0.83	0.91
1-SR	6			42	48
	5		48	6	
	4	24			
	3	24			
2-SR	6				12
	5			12	
	3		6		
3-SR	6				4
total punct. bits		168	258	342	384

For example, at the rate 0.62, instead of puncturing $168/6 = 28$ variable nodes entirely, the bit-wise pattern is spread over all 1-SR variable nodes. As the required number of punctured bits is small, puncturing variable nodes of higher levels of recoverability is not necessary. In contrast, at rates 0.71 and 0.83, where the number of required punctured bits was higher, puncturing is spread over 2-SR variable nodes as well, while at the highest rate of 0.91 all variable nodes are punctured symbol-wise.

Asymptotically optimized puncturing patterns were derived from data in Table 4 and their group distribution is shown in Table 9. In most cases, either the majority or all variable nodes are subject to puncturing.

Simulation results over the BEC are presented in Figure 29. The proposed puncturing consistently outperforms both symbol-wise and asymptotically optimized puncturing at all considered rates. There is no clear winner between symbol-wise and asymptotically

Table 9: Asymptotically optimized puncturing patterns for the non-binary LDPC code over GF(2^6).

Level	0.62	0.71	0.83	0.91
0-SR	22	0	1	0
1-SR	108	90	27	7
2-SR	11	40	34	21
3-SR	1	12	35	23
4-SR			26	14
5-SR			12	13
6-SR			6	13
7-SR			1	13
8-SR				9
9-SR				9
10-SR				9
11-SR				8
12-SR				3
total punct. bits	168	258	342	384

optimized puncturing, as at rates 0.62 and 0.71 asymptotically optimized puncturing performs better, while at 0.83 symbol-wise prevails. To achieve rate 0.83, a significant number (342) of bits needs to be punctured; therefore, choosing punctured bits according to the asymptotically optimized distribution without an effort to maintain the maximum level of recoverability low results in bad performance. Symbol-wise puncturing picks locations as to maintain the maximum level of recoverability low and, in effect, improves performance. The proposed puncturing algorithm maintains a low maximum level of recoverability and spreads the puncturing pattern over more symbols to reduce uncertainty in the decoder, which results in best performance.

Almost identical results are obtained over the AWGN channel, and they are shown in Figure 30. It is interesting to note that symbol-wise puncturing appears to be the worst choice when the overall number of punctured bits is low. At levels when heavy spreading is not likely to result in high maximum level of recoverability, puncturing all bits in a binary representation results in significant performance loss and should be avoided.

We also compare the best performing punctured non-binary LDPC codes with their binary counterparts at rates 0.62 through 0.91. The puncturing patterns for binary codes are chosen according to the grouping algorithm described in Section 2.3. The results presented

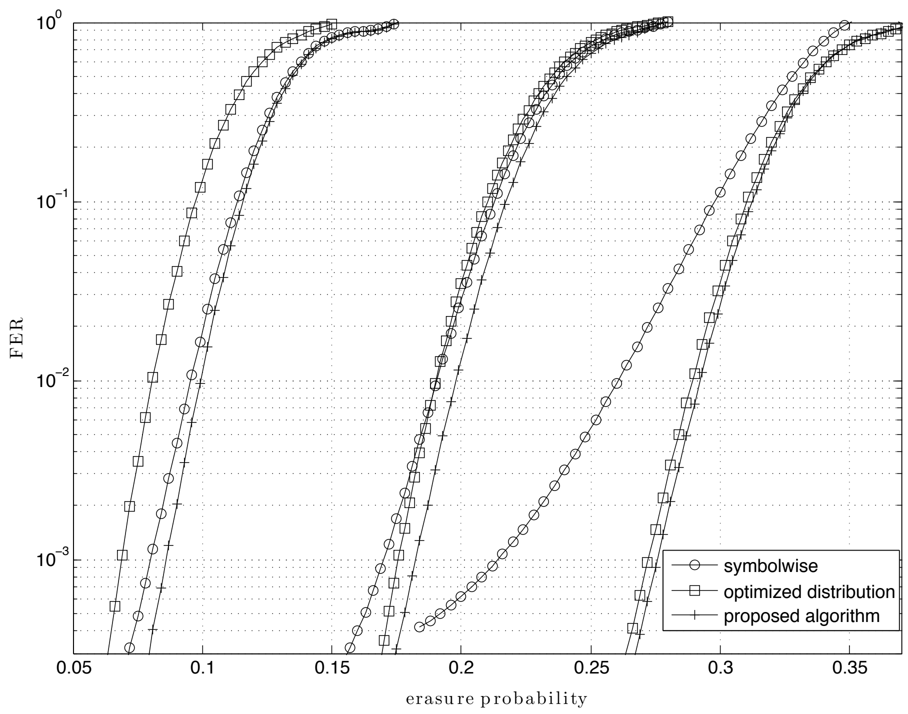

Figure 29: Performance comparison between symbol-wise, asymptotically optimized, and proposed puncturing at rates 0.62, 0.71 and 0.83 (from right to left) over the BEC channel.

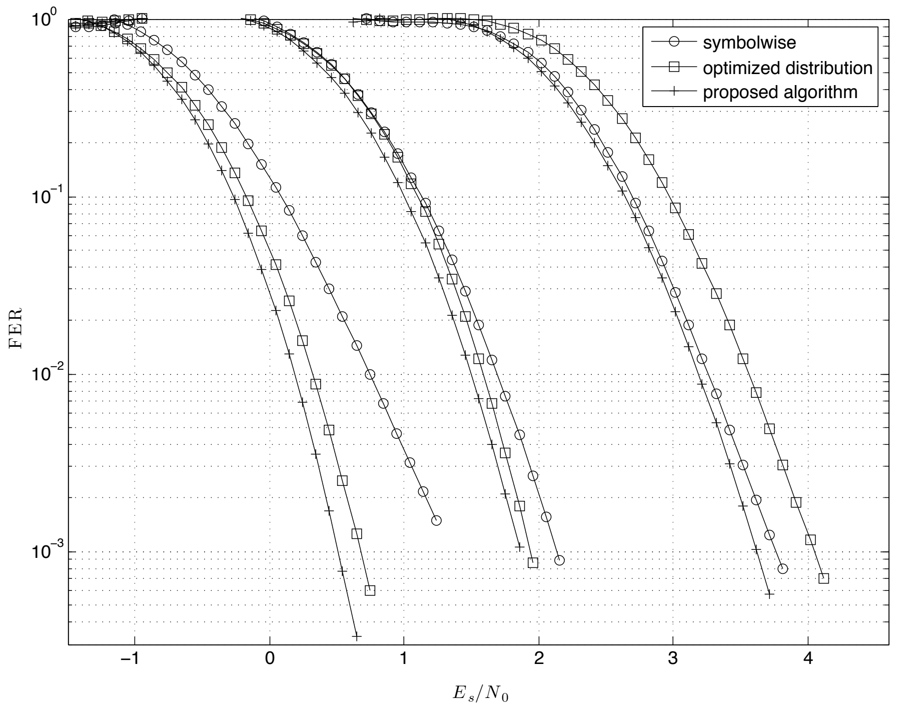

Figure 30: Performance comparison between symbol-wise, asymptotically optimized, and proposed puncturing at rates 0.62, 0.71 and 0.83 (left to right) over the AWGN channel.

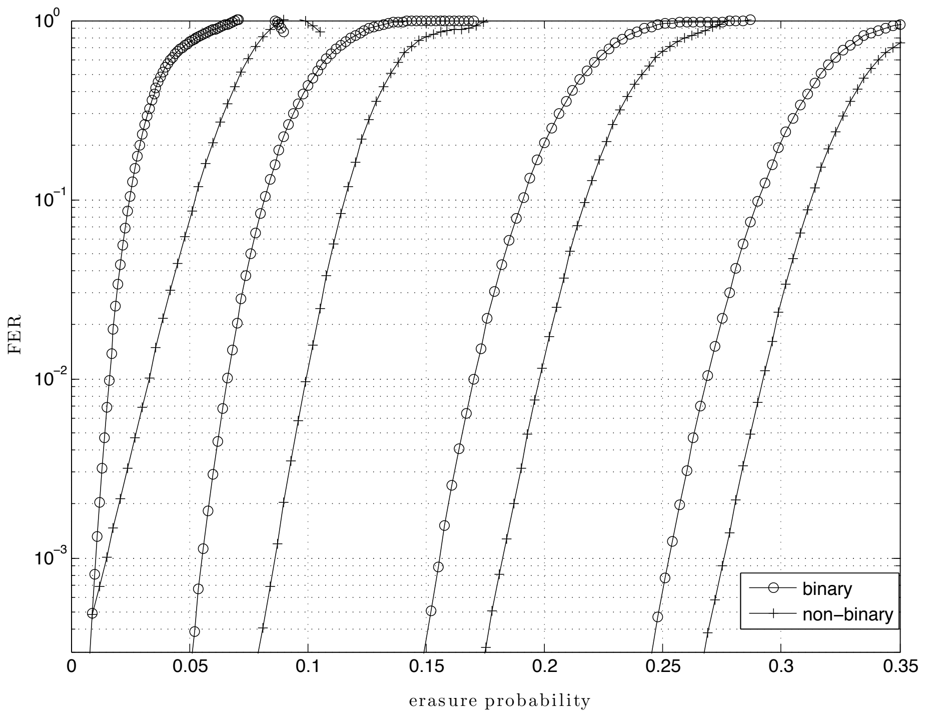

Figure 31: Performance of punctured non-binary and binary codes at rates 0.62 – 0.91 over the BEC channel.

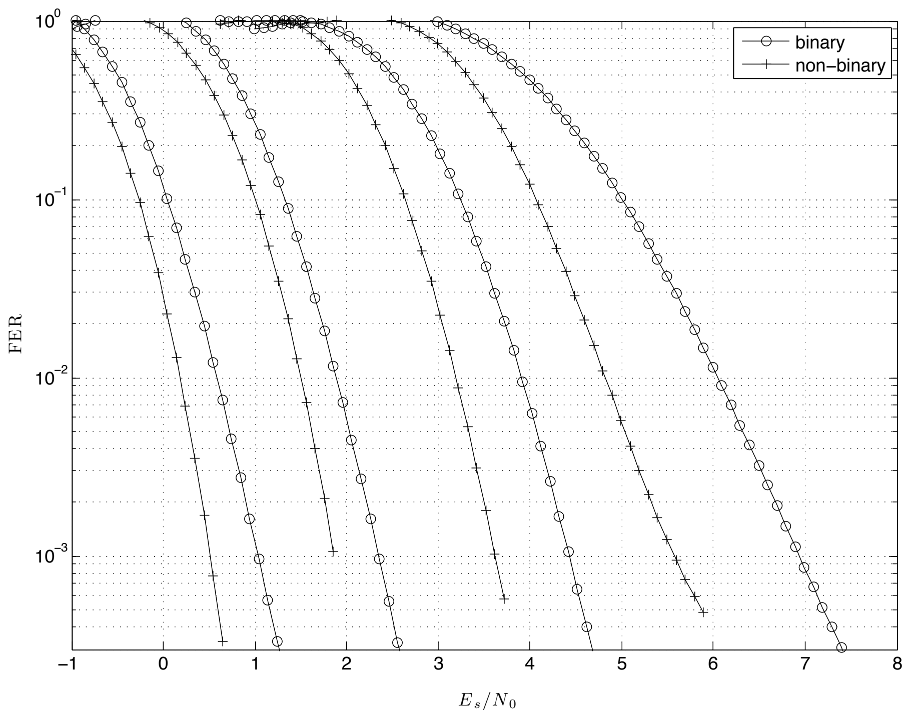

Figure 32: Performance of punctured non-binary and binary codes at rates 0.62 through 0.91 over the AWGN channel.

in Figures 31 and 32 confirm that non-binary codes outperform binary codes on both BEC and AWGN channels. In most cases the gain of non-binary codes increases with rate, except at rate 0.91, where the punctured non-binary code outperforms the binary only at low to moderate erasure probabilities over the BEC channel. Over the AWGN channel the non-binary code at rate 0.91 outperforms the binary code by a margin of roughly 1.5 dB (measured at FER of 10^{-3}).

Next we turn to shortening. As we did earlier we evaluate the proposed shortening algorithm by comparing it with symbol-wise and asymptotically optimized shortening. The shortening patterns for rates 0.23, 0.33, and 0.44 obtained with the proposed algorithm are shown in Table 10.

Table 10: The proposed shortening patterns.

codes	# of short. bits	0.23	0.33	0.44
binary	6 bits	50	35	14
GF(2^6)	6 bits			
	5 bits	16		
	4 bits	55		
	3 bits		68	
	2 bits		3	13
	1 bit			58
total short. bits		300	210	84

The performance comparison over the BEC channel is shown in Figure 33. The proposed shortening performs best at all rates, even though at rate 0.44, where the number of shortened bits is low, all approaches perform very similarly. As the rate decreases (and the number of shortened bits increases), the gains exhibited by the proposed algorithm become more pronounced. Notice the very bad performance of symbol-wise shortening at rate 0.23, which indicates that symbol-wise shortening is bad at low rates.

Similar conclusions are valid over the AWGN channel, based on plots shown in Figure 34. At the lowest rate of 0.23 the proposed shortening achives a gap of almost 1 dB over the other two approaches.

Finally, we compare how the best shortened non-binary codes compare against their binary counterparts, where shortening on the binary mother code is performed randomly

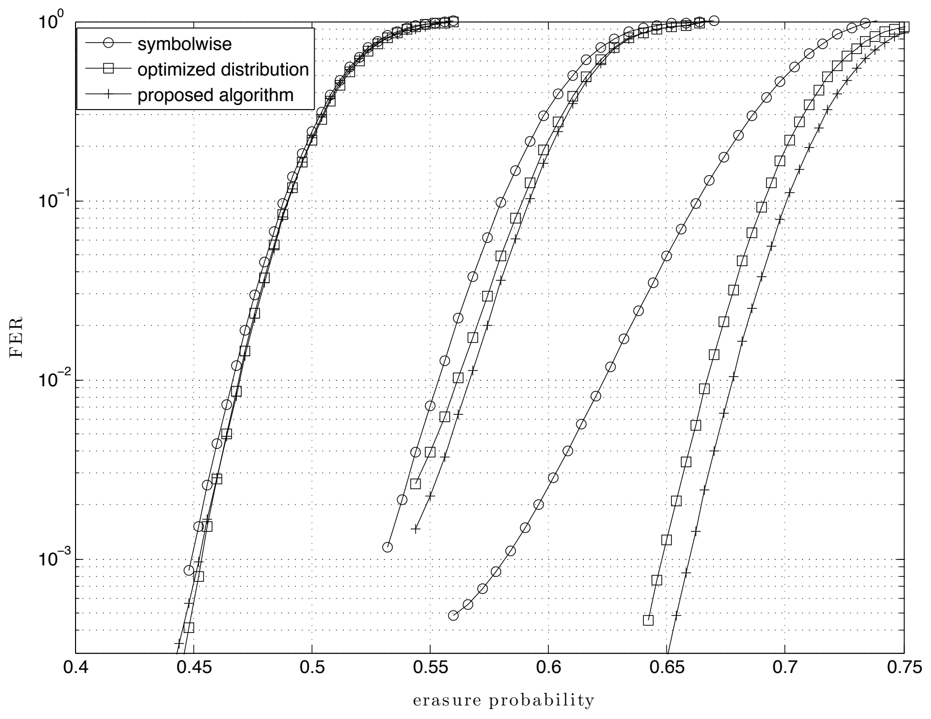

Figure 33: Performance comparison between symbol-wise, asymptotically optimized, and proposed shortening at rates 0.23, 0.33 and 0.44 (from right to left) over the BEC channel.

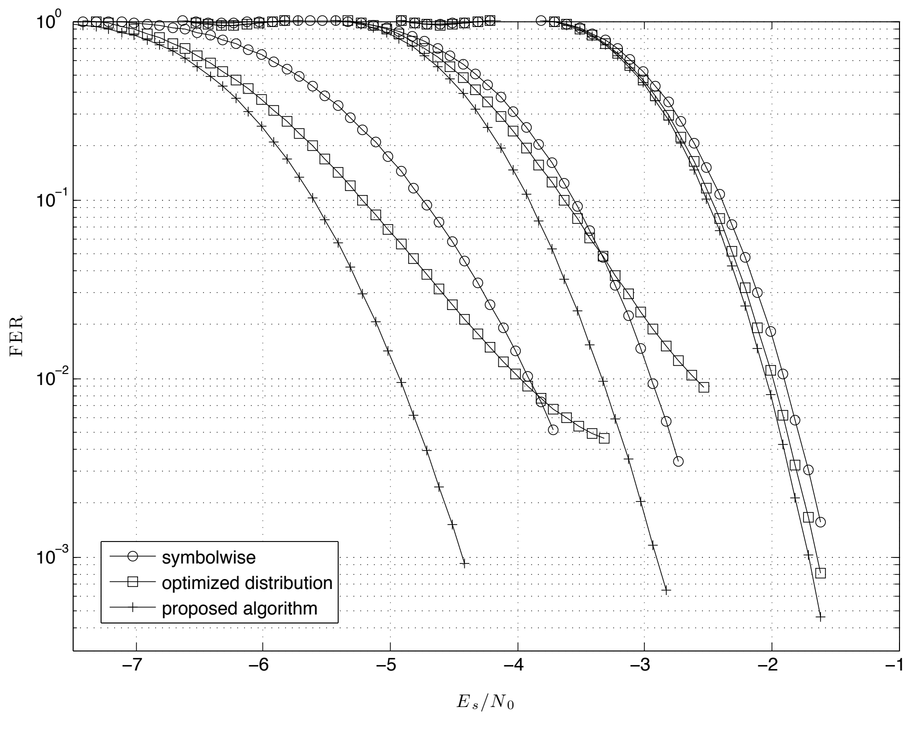

Figure 34: Performance comparison between symbol-wise, asymptotically optimized, and proposed shortening at rates 0.23, 0.33 and 0.44 (left to right) over the AWGN channel.

over the information bits. Their performance over the BEC and AWGN channels is shown in Figures 35 and 36. Again, the non-binary codes significantly outperform the binary ones and the gain increases with decreasing rate. At the lowest considered rate of 0.23 performance gap is roughly 1 dB over the AWGN channel.

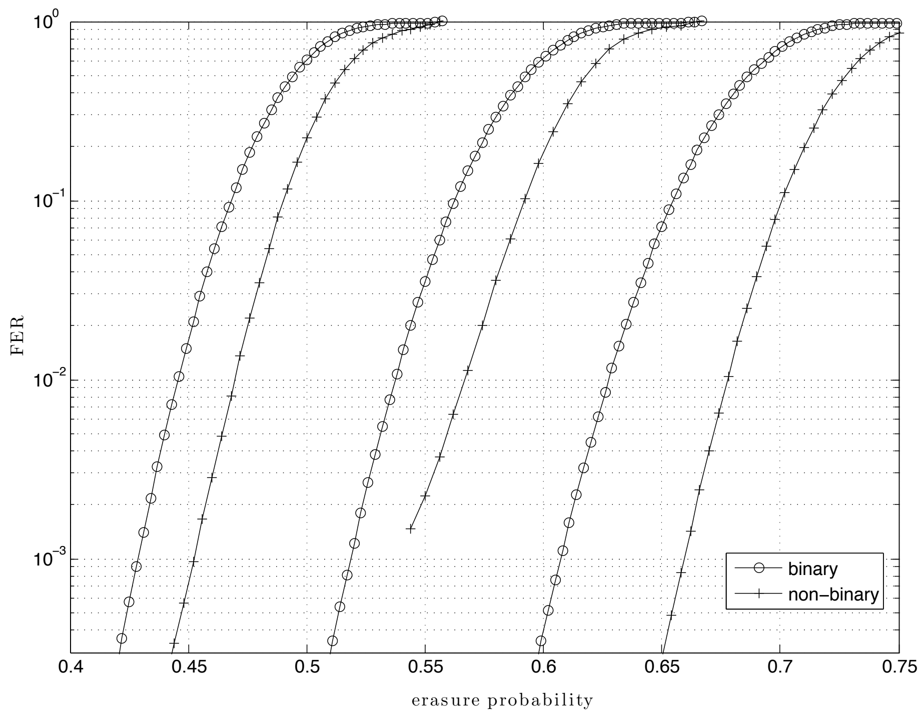

Figure 35: Performance of shortened non-binary and binary codes at rates 0.23 through 0.44 over the BEC channel.

3.6 Concluding Remarks

It was demonstrated that non-binary LDPC codes are worthy of more attention despite the higher complexity of their decoding algorithms. Punctured or shortened, they significantly outperform the binary LDPC codes at short block lengths, which are of most practical interest.

Puncturing and shortening problems attain a new degree of freedom: in addition to selecting variable nodes, puncturing/shortening depth for each variable node must be chosen. We developed a framework for asymptotic analysis of punctured and shortened non-binary LDPC codes over the BEC channel and optimized puncturing and shortening distributions in order to achieve better performance. The results show that symbol-wise puncturing

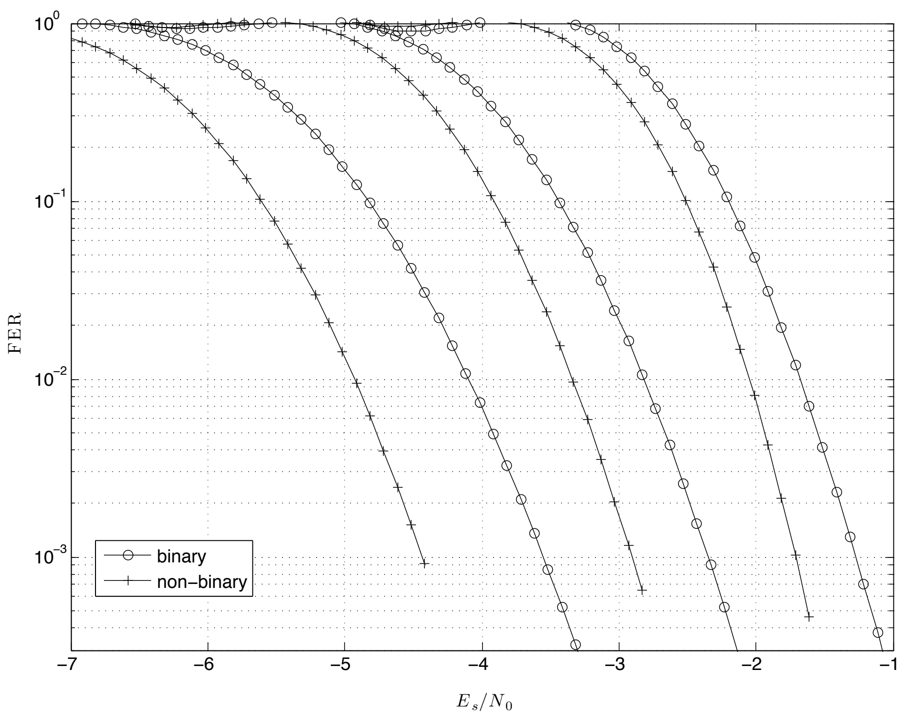

Figure 36: Performance of shortened non-binary and binary codes at rates 0.23 through 0.44 over the AWGN channel.

and shortening do not perform well, while optimized bit-wise spreading of the puncturing/shortening pattern noticeably improves performance.

Based on these conclusions we designed custom algorithms for puncturing and shortening non-binary LDPC codes with short block lengths that outperform both symbol-wise and asymptotically optimized puncturing and shortening. The algorithms are applied to specific instances of LDPC codes (as opposed to ensembles), such that structural properties of each considered Tanner graph can be taken into account when choosing punctured and shortened bits. The performance of proposed algorithms at short block lengths was verified over BEC and AWGN channels, where it was shown that the algorithms perform well and significantly outperform binary LDPC codes.

Finally, it is worth noting that rate-compatible punctured non-binary LDPC codes can be applied in distributed compression systems, where presently their binary counterparts are often the coding scheme of choice. The superior error-correction performance of non-binary LDPC codes is expected to result in increased compression gains.

CHAPTER IV

PHYSICAL LAYER SECURITY

In previous chapters we have investigated puncturing error-correction coding as a means for increasing the code rate. Here, we examine puncturing in a different setting. We focus on security and show that puncturing can be very effective for providing a high level of data security at the physical layer, which in conventional communication systems is left unsecured. After a short review of basic principles in physical layer security we proceed to show that well-performing codes for physical layer security can be designed by using puncturing.

4.1 Channel Model and Basic Notions

Consider the model depicted in Figure 37. There are three persons involved: Alice tries to send a message M to Bob, while Eve is eavesdropping on their communication. In order

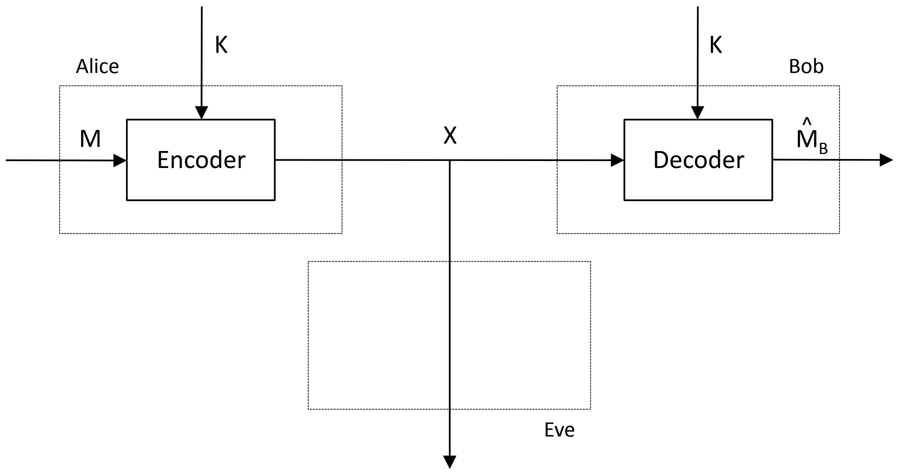

Figure 37: The classic Shannon model for secure communication.

to secure the message from Eve, Alice encodes M into the ciphertext X using the secret key K. Bob has access to the secret key K and uses it to decipher X. Eve, on the other hand, is assumed to have full knowledge about the encoding and the decoding process, but she does not have access to the secret key K and thus cannot obtain M directly. Shannon proposed [64] that the secrecy of ciphertext X be evaluated in information-theoretic terms. More specifically, he proposed that a ciphertext X be considered perfectly secure if the mutual information $I(M;X)$ between M and X equals 0. Then, we have

$$H(M/X) = H(M), \tag{59}$$

where $H(M/X)$ is the entropy of M given X, also referred to as equivocation, and $H(M)$ is the entropy of M. Eq. (59) means that Eve does not obtain any information about M by observing ciphertext X.

Shannon proved that communication in perfect secrecy in this setting is possible only if $H(K) \geq H(M)$ [64]. An example of a perfectly secure scheme is the one-time pad, for which Eve who does not have access to the secret key is provably unable to extract any information about the message. Unfortunately, the one-time pad scheme only translates the problem of sharing a message to sharing a secret key. To circumvent this difficulty, a variety of cryptographic algorithms were invented that employ shorter secret keys, but rely on unproved mathematical assumptions and limited computational resources at Eve for secrecy.

Observe that Shannon's model on Figure 37 assumes that Bob's and Eve's observations of the transmitted ciphertexts are identical. In many cases that assumption is not realistic due to the stochastic nature of many communication channels. A few decades after Shannon's work it was shown in [73, 56, 11] that information theoretically secure communication is possible exclusively by means of coding at the physical layer if Eve has a worse channel than Bob. These works assumed a slightly weaker measure of secrecy from Shannon's: instead of the absolute value of mutual information, they require that the mutual information rate

$$\frac{1}{n} I(M;X) \tag{60}$$

goes to 0, as n, the number of bits in X, goes to infinity.

Equivocation and equivocation rate at Eve are established metrics for information theoretic security [6], but they are difficult to analyze and measure on noisy coded sequences, especially at finite block lengths. That may be one of the main reasons that no practical code constructions at finite block lengths for secure communication exist at this point. The existing code constructions [69, 51] based on the insight provided from information theoretic proofs do not directly apply to continuous channels, like the Gaussian wiretap channel. To get around this problem, the BER over message bits, which is much easier to analyze and measure, is used as a measure for security. For example, if Eve observes data through a channel with BER close to 0.5 (the errors are i.i.d.), then she would not be able to extract much information about the message. It should be noted at the outset that BER is a different metric than the equivocation; therefore, this work does not address information theoretic security, but rather physical layer security. Nevertheless, it is argued that a high BER at Eve is useful and can, possibly in conjunction with standard cryptographic techniques, deliver improved resilience against eavesdropping.

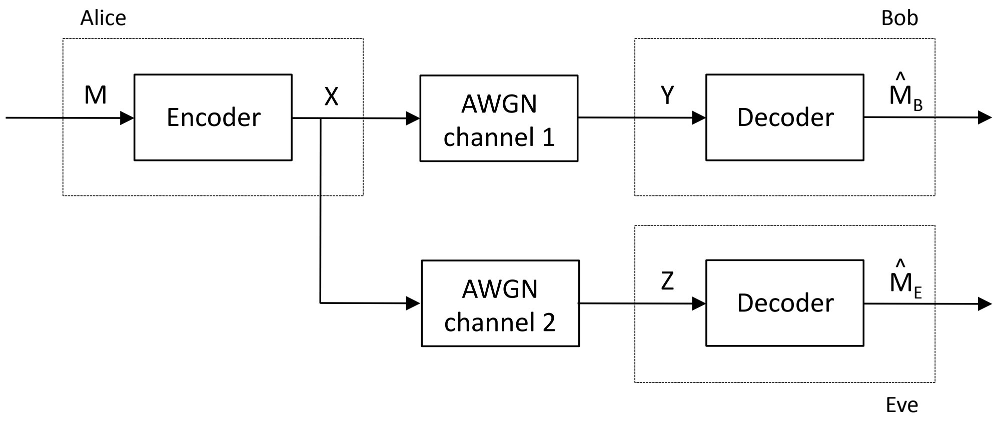

Figure 38: The Gaussian wiretap channel.

Consider the Gaussian wiretap model in Figure 38. Alice wants to transmit an s-bit message M^s to Bob. She uses an error-correction code to encode M^s to an n-bit codeword X^n and transmits it over an AWGN channel to Bob. Eve listens to the transmission over a noisier, independent AWGN channel and tries to reconstruct the message M^s. Let an average BER over the Bob's estimate $\hat{M}_B^s$ be P_e^B and let an average BER over the Eve's estimate $\hat{M}_E^s$ be P_e^E. It is desired that P_e^B be sufficiently low to ensure reliability and that P_e^E be high. If P_e^E is close to 0.5 and the errors are IID, then Eve will not be able to extract much information from the received sequence Z^n. Thus, for fixed $P_{e,\max}^B(\approx 0)$ and

$P^E_{e,\min}(\approx 0.5)$, it must hold that

(a) $P^B_e \leq P^B_{e,\max}$ (reliability),

(b) $P^E_e \geq P^E_{e,\min}$ (security).

Let $\mathrm{SNR}_{B,\min}$, or the reliability threshold, be the lowest signal-to-noise ratio for which (a) holds. This parameter has been subject to considerable scrutiny, because it constitutes one of the main performance metrics associated with error-correction codes. In this chapter we will be interested in another threshold, the security threshold $\mathrm{SNR}_{E,\max}$, which we define as the highest SNR for which (b) holds. In other words, the security threshold tells us when an error-correction code fails miserably. Rather than investigating the absolute values of these two thresholds, we will focus on their ratio $\mathrm{SNR}_{B,\min}/\mathrm{SNR}_{E,\max}$, which we call the security gap and which can alternatively be expressed in dB by taking the logarithm of the ratio. Thus, the size of the security gap in dB (see Figure 39) tells us the minimum required difference between Bob and Eve's SNRs for which secure communication in our context is possible. Note that conventional error-correction codes require large ($>$ 20 dB) security gaps when $P^E_{e,\min} > 0.4$. Our objective is to design a coding scheme that exhibits a small security gap.

The main idea is to hide data from Eve by means of puncturing. Instead of transmitting message bits, they are punctured in the encoder and must be deduced from the channel observations of the transmitted bits at the decoder. If the receiver (Eve) has a low SNR, the channel observations are expected to be very noisy; therefore, the reconstruction of punctured message bits is expected to be hard.

Binary LDPC codes are chosen as the coding scheme for two reasons: (i) their excellent error-correction performance and (ii) availability of powerful tools for asymptotical analysis of bit-wise MAP decoders both below and above the reliability threshold. Bob and Eve are assumed to use the belief propagation decoder, which is asymptotically equal to the bit-wise MAP decoder and hence very powerful. It will be shown that transmitting messages over punctured bits can significantly reduce security gaps and can thus be efficiently used for increased security of data. Security gaps as low as few dB are sufficient to force Eve to

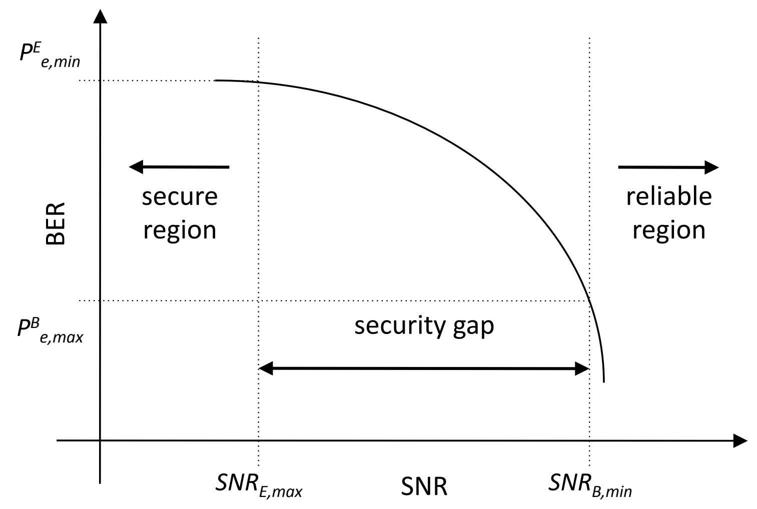

Figure 39: The security gap. A typical BER vs. SNR performance curve of an error-correction code is shown. $\text{SNR}_{B,\min}$ is the threshold for reliability (between Alice and Bob) and $\text{SNR}_{E,\max}$ is the point below which Eve has a very high error rate, typically close to 0.5. The security gap (in dB) is the SNR difference between $\text{SNR}_{B,\min} - \text{SNR}_{E,\max}$ that must be maintained between Bob and Eve in order to achieve both reliability and security constraints in conditions a) and b).

operate at BER above 0.49. The suggested coding scheme can be employed as a standalone solution or in conjunction with existing cryptographic schemes that operate on higher layers of the protocol stack.

Choosing a mother code is an important part of the design process. It will be seen that for some choices the excellent security gap performance comes with a significant increase in the reliability threshold $\text{SNR}_{B,\min}$ (compared to an unpunctured code), which in effect results in higher power consumption. Even though the main focus of this chapter is to design codes with small security gaps, we will show measures that can be taken to keep the increase in the reliability threshold low.

If an LDPC code is punctured, some of its variable nodes are not transmitted. One way of describing how a binary LDPC is punctured is by means of a puncturing distribution $\pi(x) = \sum_{i=2}^{d_v} \pi_i x^{i-1}$, where π_i denotes the fraction of variable nodes of degree i that are punctured [20]. A puncturing distribution in this form is useful for an asymptotic analysis of punctured LDPC codes, as we have seen in Chapter 3. Recall that $p^{(0)}$ denotes the fraction of all punctured bits, so that $p^{(0)} = \sum_{i=2}^{d_v} \Lambda_i \pi_i$.

Let s be the number of message bits, let k be the dimension of an LDPC code, and let n be the number of bits transmitted over the channel. Define the secrecy rate as $R_s = s/n$ and the design rate as $R_d = k/n$. Usually, in error-correction coding, the number of message bits s is equal to the dimension k of an error-correction code and thus $R_s = R_d$. However, in this chapter all messages are transmitted exclusively over the punctured bits; therefore, it may occur that $s < k$ and, in effect, $R_s < R_d$. In such cases, the unpunctured independent bit locations of a codeword are set randomly by dummy bits as depicted in Figure 40.

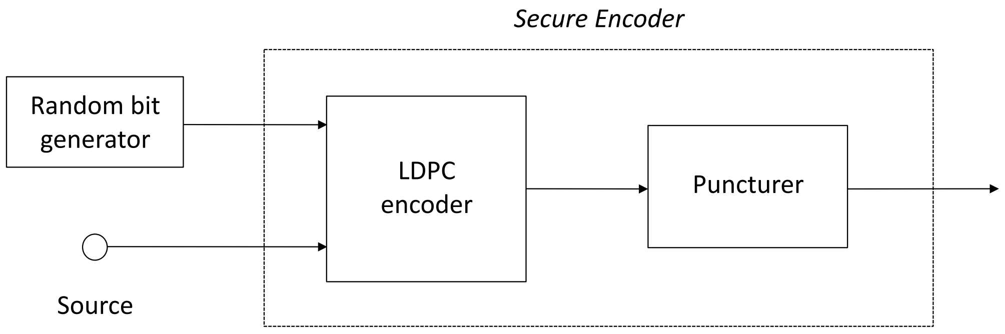

Figure 40: Block diagram of the proposed encoder.

In the following we focus on the design of secure LDPC codes in the asymptotic case and show how puncturing distributions can be optimized to significantly reduce security gaps. Some results for secure LDPC codes at finite block lengths are reported as well.

4.2 Coding for Security

The use of puncturing for improved data security is investigated in this section. The main objectives are to provide a better sense of the level of security that the proposed coding scheme delivers by analyzing the BER over (punctured) messages bits and to show how codes with small security gaps can be designed. It is assumed that the decoding algorithm is belief propagation and the analysis is asymptotic, where belief propagation decoding is equivalent to bit-wise MAP decoding [63].

4.2.1 Asymptotic Analysis

The analysis is performed with density evolution (DE) [60, 61], which is known for its accurate asymptotic analysis of the belief propagation decoder. One may be tempted to perform the analysis using the Gaussian approximation (GA) [41], a computationally less demanding alternative to DE; however, while GA was shown to perform well around and above the reliability threshold, approximation errors can be considerable when decoder is operating below the reliability threshold. To overcome this problem the analysis is performed with DE.

The calculation of the average BER performance over punctured bits with DE is now briefly discussed. According to [61] the density P_ℓ of the message from a variable node to a check node in the ℓ-th iteration for an unpunctured code is given by

$$P_\ell = P_0 \otimes \sum_{i \geq 2} \lambda_i (Q_\ell)^{\otimes(i-1)}, \tag{61}$$

where P_0 is the initial message density based on channel observations, the operator $\otimes$ denotes convolution, and Q_ℓ is the density of messages from a check node to a variable node in the ℓ-th iteration. If the code is punctured according to the puncturing distribution $\pi(x)$, the initial message density P_0^π equals

$$P_0^\pi = \sum_{i \geq 2} \lambda_i \big((1-\pi_i)P_0 + \pi_i \Delta_0\big), \tag{62}$$

where Δ_0 is the Dirac delta function $\delta(x)$. By substituting P_0 in Eq. (61) with P_0^π we obtain the density P_ℓ for punctured codes

$$\begin{aligned} P_\ell &= \sum_{i \geq 2} \lambda_i ((1-\pi_i)P_0 + \pi_i \Delta_0) \otimes (Q_\ell)^{\otimes(i-1)} \\ &= P_0 \otimes \sum_{i \geq 2} \lambda_i^{(1-\pi)} (Q_\ell)^{\otimes(i-1)} + \sum_{i \geq 2} \lambda_i^\pi \Delta_0 (Q_\ell)^{\otimes(i-1)}, \end{aligned} \tag{63}$$

where $\lambda_i^{(1-\pi)} = (1-\pi_i)\lambda_i$, $\lambda_i^\pi = \pi_i \lambda_i$. The first and second term in Eq. (63) are densities of messages emanating from unpunctured and punctured variable nodes, respectively.

The punctured variable nodes get recovered during the decoding process if they receive at least one non-zero message from neighboring check nodes. The probability that a variable node message in the ℓ-th iteration is zero can be calculated recursively using

$$P_\ell(0) = \sum_{i \geq 2} \lambda_i^\pi \big(1 - \sum_{j \geq 2} \rho_j (1 - P_{\ell-1}(0))^{j-1}\big)^{i-1},$$

and $P_0(0) = \sum_{i \geq 2} \lambda_i^{\pi}$. Assuming that punctured nodes do not form a stopping set, all nodes eventually get recovered and thus $P_\ell(0)$ tends to zero. Of most interest is the behavior of densities in the steady state, where all punctured bits are recovered.

The averaged bit error probability over the punctured variable nodes in the ℓ-th iteration is

$$P_e^{\pi,(\ell)} = \int_{-\infty}^{0^-} \Phi(x)dx + \frac{1}{2}\Phi(0),$$

where

$$\Phi(x) = \frac{1}{\sum_{i \geq 2} \Lambda_i^{\pi}} \sum_{i \geq 2} \Lambda_i^{\pi} (Q_\ell)^{\otimes i},$$

and $\Lambda_i^{\pi} \triangleq \pi_i \Lambda_i$. The bit error probability is evaluated with the discretized density evolution method [8].

Figure 41 shows the density of a check-to-variable message in the steady state, when the decoder is operating below the reliability threshold. Note that estimates for the value of punctured variable nodes depend exclusively on the incoming check node messages, as no channel value is available to the decoder. The check node density is not well approximated by the Gaussian distribution [10]. A spike is seen at 0, followed by a sharp falloff. Consequently, Eve's ability to reconstruct the message is overestimated by GA. As will be shown, the security gaps when calculated accurately with DE are even smaller than the estimates from GA.

The effectiveness of hiding message bits by means of puncturing is demonstrated on an example. A mother code of rate 1/2 with the degree distribution pair:

$$\lambda_1(x) = 0.25105x + 0.30938x^2 + 0.00104x^3 + 0.43853x^9, \tag{64}$$

$$\rho_1(x) = 0.63676x^6 + 0.36324x^7 \tag{65}$$

is punctured randomly according to

$$\pi(x) = 0.4x + 0.4x^2 + 0.4x^3 + 0.4x^9, \tag{66}$$

The overall fraction of punctured bits $p^{(0)}$ is 0.4 and all punctured bits are assumed to carry messages; therefore, $R_s = p^{(0)}/(1 - p^{(0)}) = 2/3$. The unpunctured mother code has

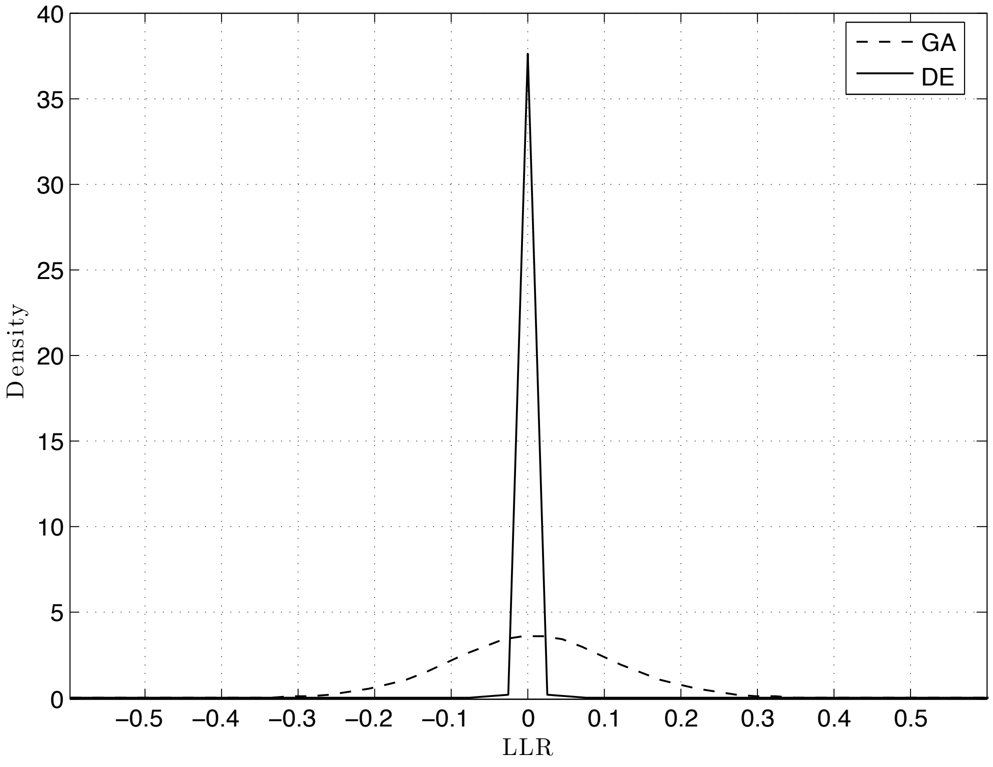

Figure 41: Density of a check-to-variable node message obtained by GA and DE when the decoder operates below the reliability threshold.

$R_d = 1/2$, while the punctured code has $R_d = k/(2k(1-p^{(0)})) = 5/6$. Since R_s and R_d are not equal for the punctured code, we must randomly set $k - s = n/6$ variable nodes in the encoder.

For comparison, we consider an unpunctured LDPC code with $R_s = R_d = 2/3$ and degree distribution pair[1]

$$\lambda_2(x) = 0.17599x + 0.40223x^2 + 0.42178x^9 \tag{67}$$

$$\rho_2(x) = 0.61540x^{10} + 0.38460x^{11}. \tag{68}$$

Eve's BER over message bits for these two codes are shown in Figure 42. Observe Eve's BER as her SNR decreases from the reliability threshold. If the message bits are punctured, Eve's BER increases much faster with the growing gap to the reliability threshold than it does when the messages are transmitted. For instance, for $P_{e,\min}^{\mathrm{E}}$ set to 0.40, 0.45 and 0.49,

[1]Most degree distribution pairs in this Chapter were obtained at http://lthcwww.epfl.ch/research/ldpcopt/

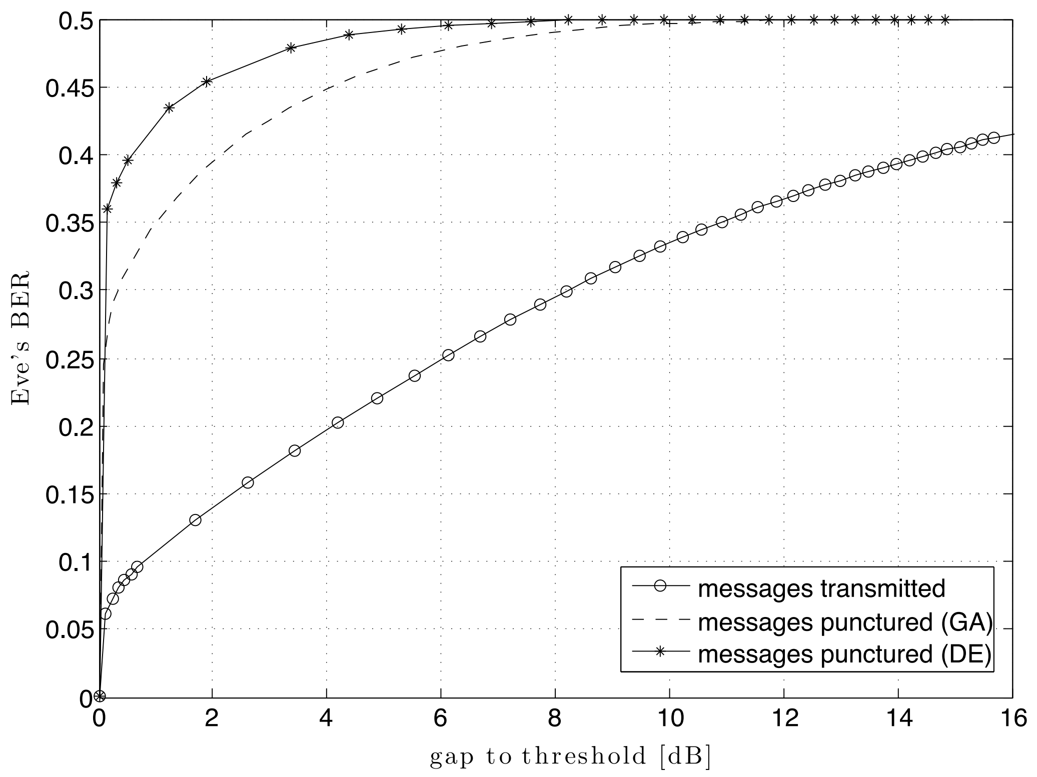

Figure 42: Eve's BER performance when operating below the reliability threshold $\mathrm{SNR}_{B,\min}$ and message bits are (i) transmitted and (ii) punctured.

the security gaps amount to 0.6 dB, 1.8 dB and 4.1 dB, respectively. In contrast, if the message bits are transmitted over the channel, the security gaps are considerably larger at 14 dB, 20 dB and 34 dB, respectively. These results manifest the benefits of protecting the message bits by puncturing. They show that small security gaps are sufficient to force Eve to operate at relatively high BERs even if she has the capability of using a bit-wise MAP decoder.

While not apparent from Figure 42, it must be noted that the increased security is leveraged at the expense of an increased reliability threshold $\mathrm{SNR}_{B,\min}$. As we mentioned earlier, some of the independent variable nodes need to be set randomly in Bob's encoder when $R_s < R_d$; thus, we use an LDPC code with design rate R_d, but messages are transmitted at a lower rate R_s. Consequently, an unpunctured code with all independent bits used for messages is expected to have a lower reliability threshold. In our example, the reliability threshold $\mathrm{SNR}_{B,\min}$ for the unpunctured code is -0.48 dB, whereas for the punctured code it is 2.28 dB; an increase of 2.76 dB. In the following, we refer to the difference in $\text{SNR}_{B,\min}$ between a punctured code and an unpunctured code with the same R_s as SNR loss.

4.2.2 Optimized Puncturing Distributions

A natural question at this point is whether lower security gaps are achievable by optimizing the puncturing distribution for security instead of using a random one. To get an answer, a mother code with the degree distribution pair in Eqs. (64) and (65) is punctured in two different manners: (i) randomly, and (ii) according to a puncturing distribution optimized to minimize the security gap and obtained with differential evolution [67]. The optimized puncturing distributions are given in Table 11.

Table 11: Puncturing distributions optimized for security. $P^E_{e,\min}$ was set to 0.49.

$p^{(0)}$	0.10	0.25	0.33	0.40
secrecy rate R_s	0.11	0.33	0.50	0.67
π_2	0.1073	0.3105	0.4930	0.5519
π_3	0.1310	0.0010	0.0004	0.1378
π_4	0.8703	0.0121	0.1170	0.4765
π_{10}	0.0015	0.6639	0.6400	0.5814
security gap [dB]	4.346	4.086	4.034	4.386

The performance comparison between random and optimized puncturing is shown in Figure 43, where 4 different puncturing fractions are considered: 0.10, 0.25, 0.33, and 0.40, which correspond to secrecy rates 0.11, 0.33, 0.50, and 0.67, respectively. SNR loss at each considered rate is depicted as well.

The benefit of optimized puncturing distributions for security is most pronounced at high secrecy rates. The gains over random puncturing of up to 0.4 dB were achieved, a considerable improvement at asymptotic block lengths. The security gap can be reduced at expense of lower secrecy rate; however, reducing the secrecy rate below 0.43 in this case may not be reasonable due to negative effects on the security gap and SNR loss.

4.2.3 Reducing SNR Loss

The SNR loss translates into higher power consumption at the transmitter and here we describe how the SNR loss can be reduced for systems subject to stringent power consumption

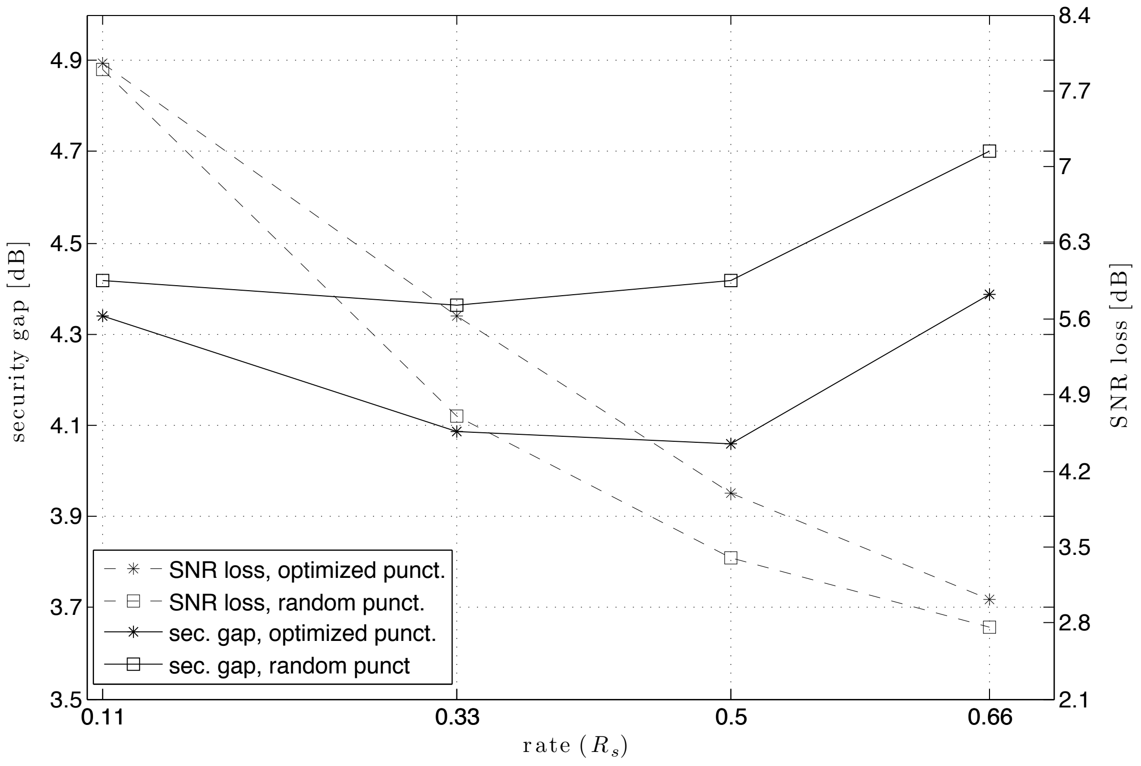

Figure 43: The performance comparison between random and optimized puncturing. The probability $P_{e,\min}^{E} = 0.49$.

constraints.

Two main factors that cause the SNR loss are puncturing loss and rate loss. Puncturing loss occurs due to inferior performance of punctured codes as compared to unpunctured codes, as observed in [20, 58, 39], but can generally be kept relatively low (< 1 dB). On the other hand, rate loss occurs when $R_s < R_d$, that is when the number of message bits transmitted per codeword is smaller than the dimension of a code.

While empirical evidence in [20, 58] suggests that puncturing loss cannot be prevented, rate loss can be eliminated completely by codes with $R_s = R_d$. For this purpose we design secure LDPC codes by using mother codes with design rates 0.10, 0.25, 0.33, and 0.40, where all independent variable nodes are punctured in order to achieve equality $R_s = R_d$. Since neither R_s nor R_d should be higher than 1 the rate of the mother code must not exceed 0.5.

The degree distribution pairs for these mother codes (listed in the same order as above) are

$$\begin{aligned}
\lambda_3(x) &= 0.551251x + 0.203119x^2 + 0.0917565x^4 + 0.00428x^6 + 0.01705x^7 + \\
&\quad 0.09970x^8 + 0.03284x^9, & (69)\\
\rho_3(x) &= x^2 & (70)
\end{aligned}$$

$$\begin{aligned}
\lambda_4(x) &= 0.38961x + 0.199689x^2 + 0.110605x^3 + 0.00971174x^4 + 0.290384x^9, & (71)\\
\rho_4(x) &= 0.8x^3 + 0.2x^4 & (72)
\end{aligned}$$

$$\begin{aligned}
\lambda_5(x) &= 0.334539x + 0.242082x^2 + 0.054702x^3 + 0.0000052x^4 + 0.368671x^9, & (73)\\
\rho_5(x) &= x^4 & (74)
\end{aligned}$$

$$\begin{aligned}
\lambda_6(x) &= 0.29445x + 0.257133x^2 + 0.448417x^9, & (75)\\
\rho_6(x) &= x^5 & (76)\\
& & (77)
\end{aligned}$$

We optimize puncturing distributions to minimize the security gap, while making sure that R_s equals R_d. The puncturing fractions $p^{(0)}$ for above listed degree distribution pairs are fixed at 0.10, 0.25, 0.33, and 0.40, respectively, and the corresponding secrecy rates R_s are 0.11, 0.33, 0.50, and 0.67. The new optimized puncturing distributions are given in Table 12.

The new codes are compared in Figure 44 with the punctured codes from Section 4.2.2, which were derived from a mother code with rate 0.5. Note that for those codes $R_s < R_d$ and rate loss is non-zero. It is seen that the SNR loss is significantly reduced when the code does not suffer rate loss. At most rates the SNR loss is kept below 1 dB, while with $R_s < R_d$ codes it ranged from 4 dB to 8 dB. For codes with no rate loss, the SNR loss increases with the increasing secrecy rate. These losses are incurred by puncturing and increase with the increasing fraction of punctured bits $p^{(0)}$, much like it was observed in [20, 58].

Table 12: Optimized puncturing distributions for $R_s = R_d$ codes and $P^E_{e,\min} = 0.49$.

$p^{(0)}$	0.10	0.25	0.33	0.40
secrecy rate R_s	0.11	0.33	0.50	0.67
π_2	0.1370	0.2619	0.3589	0.5105
π_3	0.0015	0.0014	0.0063	0.0013
π_4	—	0.0002	0.0818	—
π_5	0.0003	0.0424	0.3594	—
π_7	0.0093	—	—	—
π_8	0.0021	—	—	—
π_9	0.0003	—	—	—
π_{10}	0.0013	0.9916	0.9993	0.7992
security gap [dB]	9.992	6.644	4.906	4.2193

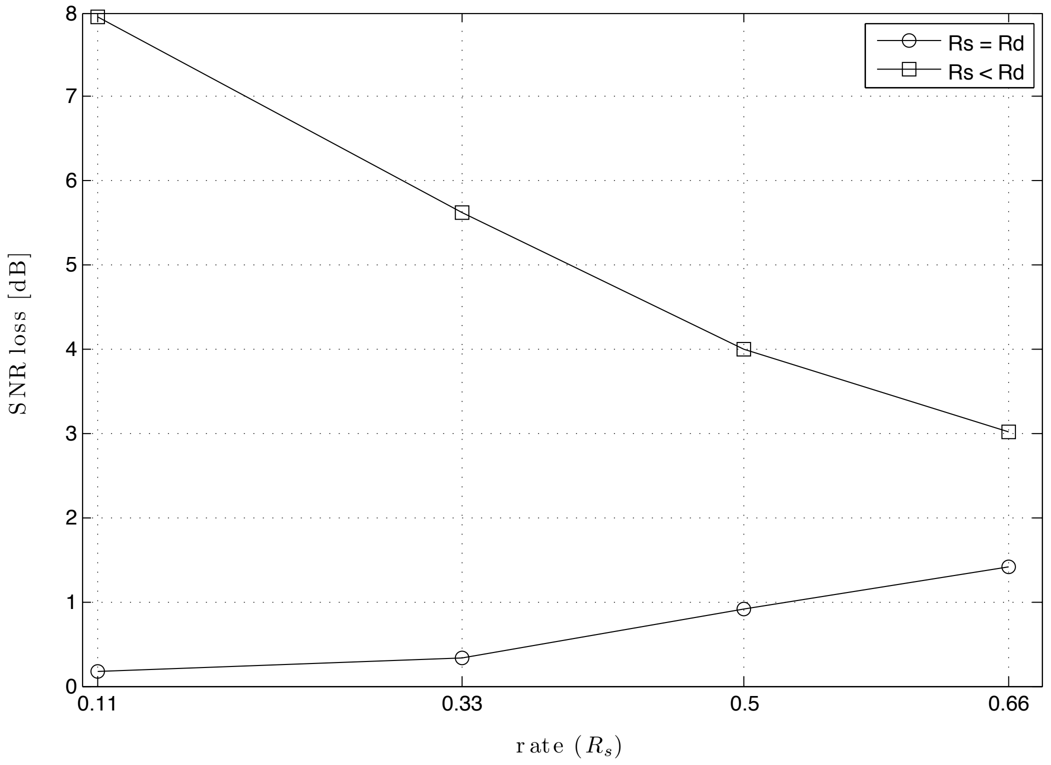

Figure 44: SNR loss comparison between codes with $R_s = R_d$ and $R_s < R_d$.

The $R_s = R_d$ codes are superior in terms of power consumption, while the $R_s < R_d$ codes have smaller security gaps for most considered rates, as is shown in Figure 45, where all security gaps are measured for $P^E_{e,\min} = 0.49$. These results indicate there is a trade-off between the SNR loss and the security gap.

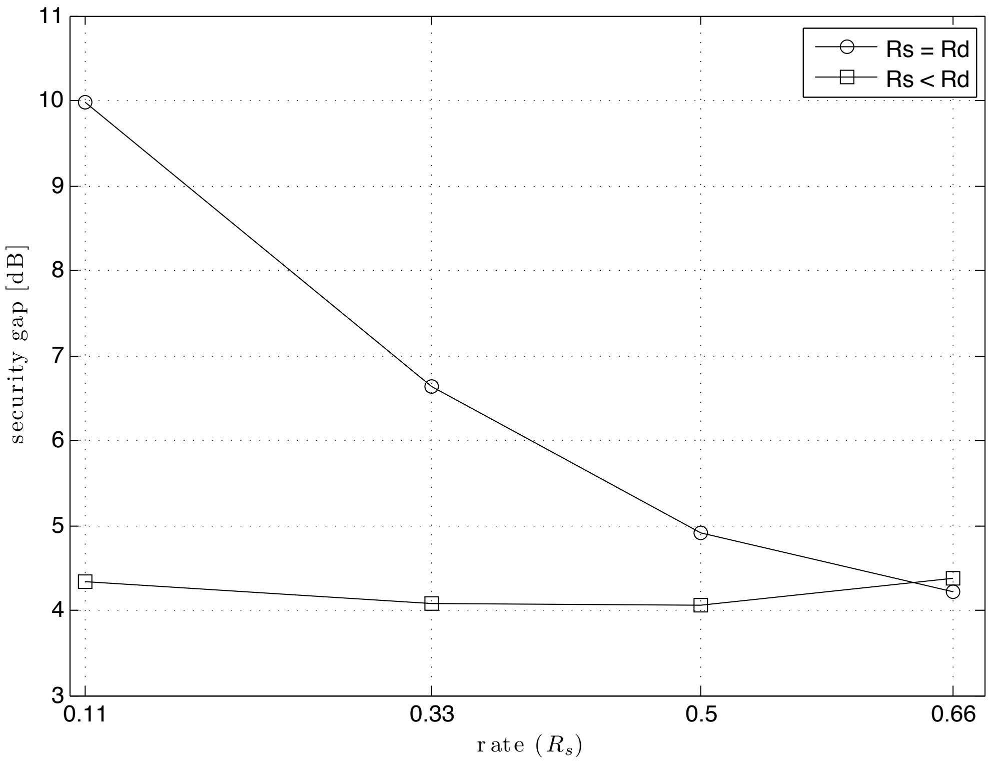

Figure 45: Security gap comparison between $R_s = R_d$ and $R_s < R_d$ codes.

4.2.4 Finite Block Lengths

The performance of codes from Figure 42 is evaluated for random puncturing at finite block lengths and presented in Figure 46, where the number of message bits is 1576, the number of transmitted bits in each block is 2364 and $P_{e,\max}^B$ is set to 10^{-5}. With random puncturing, security gaps as low as a few dB are attainable for $P_{e,\min}^E = 0.4$. Since at finite block lengths Tanner graphs of LDPC codes usually have cycles, belief propagation decoding does not yield exact bit-wise MAP probabilities of codeword bits. Nevertheless, the belief propagation was shown to exhibit excellent performance.

4.3 Efficient Encoding

It is shown in [62] that encoding in linear time is possible for many LDPC codes; but the assumption is that any variable node can be chosen to carry the message, which is not true for the specific case considered here. This section addresses the encoding problem and shows that efficient encoding is possible if messages are transmitted in the proposed manner.

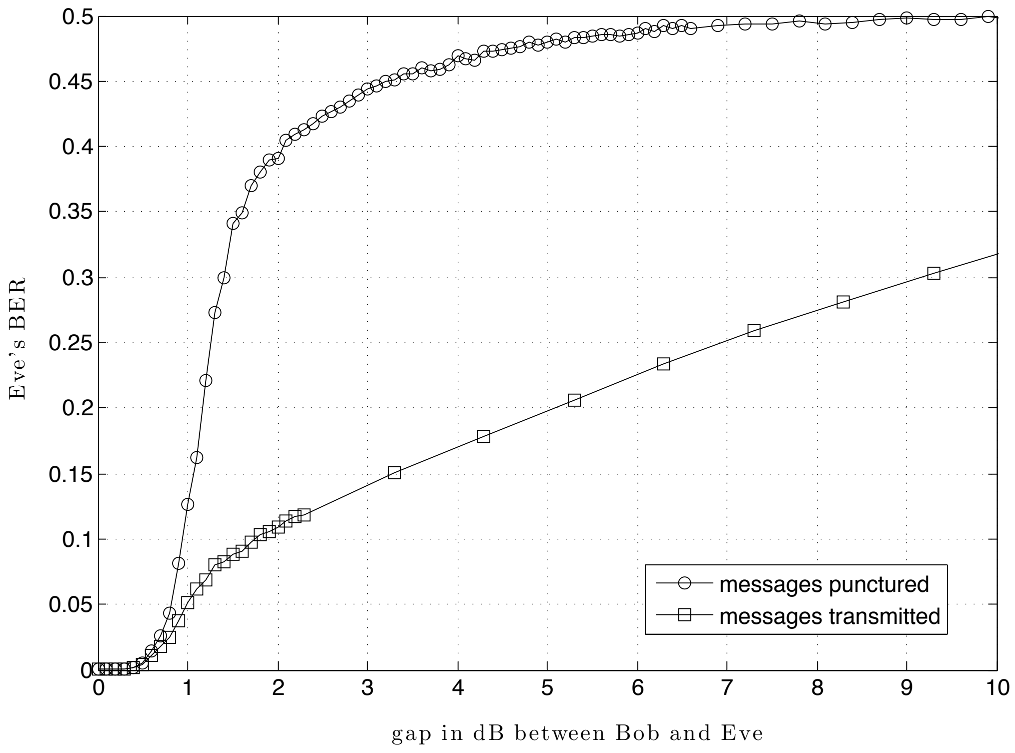

Figure 46: BER vs. security gap at block length 2364.

The main result in [62] states that any LDPC code whose matrix can be brought into the form shown in Figure 47 by means of row and column permutations is encodable with complexity $O(n + g^2)$, where g is referred to as gap. Efficient encoding is very closely

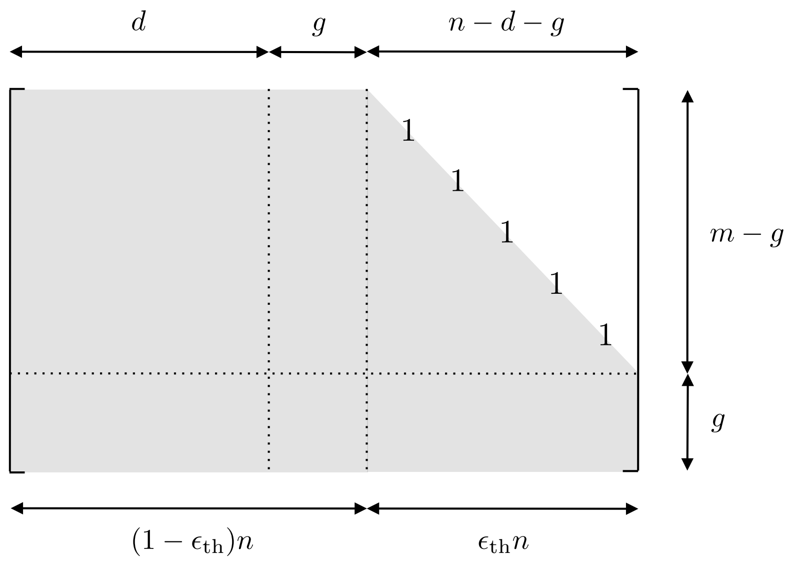

Figure 47: Lower-triangular parity-check metrix suitable for efficient encoding.

related to code's ability to recover from erasures over the BEC. For an LDPC code with the degree distribution pair $(\lambda(x), \rho(x))$, let $\epsilon_{\text{th}}(\lambda, \rho)$ denote its BEC threshold and $r(\lambda, \rho) = 1 - (\int_0^1 \rho(x)\,\mathrm{d}x / \int_0^1 \lambda(x)\,\mathrm{d}x)$ its rate. It was shown that using their greedy algorithm A,

$$g = \big(1 - r(\lambda, \rho) - \epsilon_{\text{th}}(\lambda, \rho)\big)n \tag{78}$$

is achievable asymptotically for large n. Sometimes, smaller gaps are attainable if triangulation is performed on the transpose of the parity check matrix. In that case the attainable gap becomes

$$g^{(T)} = \frac{1 - \epsilon_{\text{th}}(\rho, \lambda)}{1 - r(\rho, \lambda)} n. \tag{79}$$

Note that in a transposed parity-check matrix the degree distributions of variable and check nodes are interchanged.

If puncturing is random the greedy algorithm A from [62] can be applied directly, and then, if the mother code's BEC threshold is close to capacity, the achievable gap g is small and the code is efficiently encodable. On the other hand, if puncturing is not random the analysis from [62] has to be modified, as the choice of variable nodes that carry messages is not random anymore.

Assume that messages are transmitted over punctured nodes and the puncturing pattern is given by $\pi(x)$. This pattern restricts parity-check matrix columns that may be permuted to achieve the triangular structure to only unpunctured variable nodes. The methods from [62] must be applied to the residual parity-check matrix left after deleting the columns that correspond to punctured bits. Let $(\lambda_{\text{res}}(x), \rho_{\text{res}}(x))$ be the degree distribution pair of the residual parity-check matrix and $\Lambda_{\text{res},i}$ the fraction of variable nodes of degree i. Then

$$\Lambda_{\text{res},i} = \frac{(1 - \pi_i)\Lambda_i}{\sum_{j=2}^{d_v}(1 - \pi_j)\Lambda_j} \tag{80}$$

and

$$\lambda_{\text{res},i} = \frac{i\Lambda_{\text{res},i}}{\sum_{j=2}^{d_v} j\Lambda_{\text{res},j}}. \tag{81}$$

The average check node degree after puncturing is

$$d_{c,\text{res}} = \frac{1 - \sum_{i=2}^{d_v} \pi_i \lambda_i}{(1 - r(\lambda, \rho)) \sum_{i=2}^{d_v} \lambda_i / i}. \tag{82}$$

In general, the residual check node distribution is not deterministic. This difficulty is circumvented by assuming that the distribution is close to uniform and comprised of two terms at most. Then, the residual check node distribution from the node perspective is given by

$$P_{\text{res}}(x) = \big(1 - \lfloor d_{c,\text{res}} \rfloor\big) \cdot x^{\lfloor d_{c,\text{res}} \rfloor} + \big(d_{c,\text{res}} - \lfloor d_{c,\text{res}} \rfloor\big) \cdot x^{\lfloor d_{c,\text{res}} \rfloor + 1}, \tag{83}$$

while from the edge perspective it is

$$\rho_{\text{res}}(x) = \frac{P'_{\text{res}}(x)}{P'_{\text{res}}(1)}, \tag{84}$$

where $P'_{\text{res}}(x)$ denotes the derivative of $P_{\text{res}}(x)$. With the residual degree distribution the attainable gaps g and $g^{(T)}$ are given by

$$g = \big(1 - r(\lambda_{\text{res}}, \rho_{\text{res}}) - \epsilon_{\text{th}}(\lambda_{\text{res}}, \rho_{\text{res}})\big)(1 - p^{(0)})n \tag{85}$$

and

$$g^{(T)} = \frac{1 - \epsilon_{\text{th}}(\rho_{\text{res}}, \lambda_{\text{res}})}{1 - r(\rho_{\text{res}}, \lambda_{\text{res}})}(1 - p^{(0)})n. \tag{86}$$

Table 13 lists the encoding gaps for optimized puncturing distributions from Table 11. The asymptotic results for $g^{(T)}$ were verified using a mother code with the block length 100000 bits. These empirical results are close to the analytical ones, and the slight deviation can be attributed to the crude assumption of uniformity of the residual check node degree distribution.

Table 13: Asymptotic and empirical attainable gaps.

$p^{(0)}$	g	$g^{(T)}$	empirical
0	$0.0298n$	0	$10^{-5}n$
0.1	$0.0321n$	0	$10^{-5}n$
0.2	$0.0304n$	0	$10^{-5}n$
0.3	$0.0379n$	$0.0036n$	$0.0089n$
0.4	$0.0553n$	$0.0137n$	$0.0162n$

4.4 System Aspects

The proposed coding scheme operates on the physical layer of the protocol stack and can be viewed as a first step towards a somewhat unconventional bottom-up architecture for secure communications. In traditional cryptography the physical layer merely provides the higher layers with a virtually error-free channel abstraction, and it is argued that stronger levels of secrecy are achievable by redesigning the channel coding modules according to the aforementioned security metrics. Intuitively it is to be expected that a noisy ciphertext is more difficult to break than its error-free counterpart. At BER close to 0.5 there is little correlation left between the signals observed by the eavesdropper and the original message. To break the cipher she would have to guess both the key and the random error sequence introduced by the channel, which leads to a significant increase in the search space while performing cryptanalysis.

Ultimately, a wireless link should be at least as difficult to eavesdrop as an Ethernet cable, so that the attacker would have to gain physical access to the channel at very close proximity to be able to acquire information bearing signals. Even if this threshold is exceeded, the attacker would still have to break hard cryptographic primitives. In other words, the proposed coding scheme does not necessarily replace cryptography, yet it adds one more layer of protection that is targeted at the lower and possibly most vulnerable stage of wireless devices.

While wiretap codes, whose practical construction is still elusive for most cases of interest, would use part of the rate to confuse the eavesdropper and to achieve information-theoretic security (at least asymptotically), the proposed LDPC codes can be readily implemented to induce higher BER at the eavesdropper with a controlled reduction of the rate of communication.

One could argue that cryptographic primitives such as the Advanced Encryption Standard (AES) are designed for the worst case in which the eavesdropper acquires an error-free ciphertext. In applications, such as RFID systems and wireless sensor networks, where strong ciphers like AES are too costly computationally, the proposed codes for security can be combined with lightweight ciphers, while still ensuring sufficient levels of confidentiality. Thus, joint design of channel codes and cryptographic primitives emerges as a worthwhile line of research [25, 26].

4.5 Concluding Remarks

In this chapter an alternate approach to the design of error-correction codes for the Gaussian wiretap channel is explored. Instead of equivocation, which is believed to be challenging to analyze and measure, we choose BER over message bits at the eavesdropper as the security metric given that a bit-wise MAP decoder is available. We define the security gap as a measure of separation between the reliability and security thresholds of an error-correction code and propose codes that minimize the security gap. These codes exhibit a very sharp increase in BER toward 0.5 as the signal deteriorates, which prevents eavesdroppers from extracting the message even if wiretapped signal is only slightly weaker than that of the intended receiver.

In proposed punctured LDPC codes all message bits are communicated exclusively over punctured bits. The asymptotic analysis shows this to be effective as it yields security gaps as small as a few dB, which is a significant improvement over conventional error-correction codes or uncoded transmission. It is worth noting that puncturing the messages creates a non-systematic code. We conjecture that most non-systematic error-correction codes would exhibit good security gap performance, but asymptotic analysis of bit-wise MAP decoders is hard for many types of error-correction codes. The benefit of the proposed LDPC coding scheme is in that it can be effectively analyzed using existing tools and therefore reliable assessment can be made about the eavesdropper's performance above and below the reliability threshold.

The proposed codes are encodable in linear time and can be effectively applied to practical, finite-block length systems. While in some cases such codes can be used as a stand-alone security solution, it is perhaps more prudent to view them as an addition to the existing layered approach, which adds security to the traditionally unsecured physical layer.

CHAPTER V

COMPRESSION OF DATA ENCRYPTED WITH BLOCK CIPHERS

This chapter examines the interesting problem of compressing encrypted data that is solved using error-correction codes; in a slight departure from previous chapters puncturing finds no application here.

Traditionally in communication systems, data from a source are first compressed and then encrypted before transmission over a channel to the receiver. While in many cases this approach is befitting, there exist scenarios where there is a need to reverse the order in which data compression and encryption are performed. For example, consider a network of low-cost sensor nodes that transmit sensitive information over the internet. The sensor nodes need to encrypt data to hide it from potential eavesdroppers, but they may not be able to perform compression as that would require additional hardware and thus higher implementation cost. On the other hand, the network operator that is responsible for transfer of data to the recipient wants to compress the data to maximize the utilization of its resources. It is important to note that the network operator is not trusted and hence does not have access to the key used for encryption and decryption. If it had the key, it could simply decrypt data, compress and encrypt again.

We focus on compression of data encrypted with block ciphers, such as the Advanced Encryption Standard (AES) [54] and Data Encryption Standard (DES) [55], which operate on inputs of fixed length and serve as important building blocks that can be used to construct secure encryption schemes.

For a fixed key a block cipher is a bijection, therefore input and output have the same entropy. It follows that it is theoretically possible to compress the source to the same level as before encryption; however, in practice, encrypted data appears to be random and the conventional compression techniques do not yield desirable results. It was long believed that encrypted data are practically incompressible. A surprising paper [31] breaks that paradigm and shows that the problem of compressing one-time pad encrypted data translates into the problem of compressing correlated sources, which was solved by Slepian and Wolf in [66] and for which practical and efficient codes are known. Compression is practically achievable due to a simple symbol-wise correlation between the key (one-time pad) and the encrypted message. However, when the correlation is more complex, as in the case of block ciphers, the approach to Slepian-Wolf coding utilized in [31] is not directly applicable.

Here we investigate if data encrypted with block ciphers can be compressed without access to the key. We show that block ciphers in conjunction with the most commonly used chaining modes in practice (e.g., [34, 15]) are practically compressible for some types of sources. To our knowledge this is the first work to show that substantial compression gains can be achieved for cryptographic algorithms like AES and DES when they are used in non-stream modes. In particular, this work offers a solution to the open problem formulated in [32, Sec. 3.3].

5.1 Preliminaries

We begin with a standard formal definition of an encryption scheme as stated in [33]. A private-key encryption scheme is a triple of algorithms (Gen, Enc, Dec), where Gen is a probabilistic algorithm that outputs a key K chosen according to some distribution that is determined by the scheme; the encryption algorithm Enc takes as input a key K and a plaintext message X and outputs a ciphertext $Enc_K(X)$; the decryption algorithm Dec takes as input a key K and a ciphertext $Enc_K(X)$ and outputs a plaintext $Dec_K(Enc_K(X)) = X$. It is required that for every key K output by Gen and every plaintext X, we have $Dec_K(Enc_K(X)) = X$.

In private-key encryption schemes of concern to us in this chapter, the same key is used for encryption and decryption algorithms. Private-key encryption schemes are divided in two categories: stream ciphers and block ciphers. Stream ciphers encrypt plaintext one symbol at a time, typically by summing it with a key (XOR operation for binary alphabets). In contrast, block ciphers accomplish encryption by means of nonlinear mappings on input blocks of fixed length; common examples are AES and DES. Block ciphers are typically not used as a stand-alone encryption procedure; instead, they are combined to work on variable-length data using composition mechanisms known as chaining modes or modes of operation, as described in Section 5.2.

We proceed with a formulation of the source coding problem with decoder side-information illustrated in Figure 48. Consider random variables X (termed the source) and S (termed the side-information), both over a finite alphabet and with a joint probability distribution P_{XS}. Let a sequence of independent n realizations of (X, S) be denoted by $\{X_i, S_i\}_{i=1}^n$.

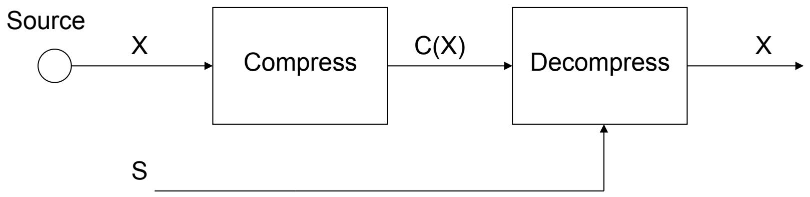

Figure 48: Lossless source coding with decoder side-information.

The problem at hand is to losslessly encode $\{X_i\}_{i=1}^n$, with $\{S_i\}_{i=1}^n$ known only to the decoder. In [66], Slepian and Wolf showed that for sufficiently large block length n, this can be done at rates arbitrarily close to the conditional entropy $H(X|S)$. Practical Slepian-Wolf coding schemes use constructions based on good linear error-correcting codes [1, 18, 48].

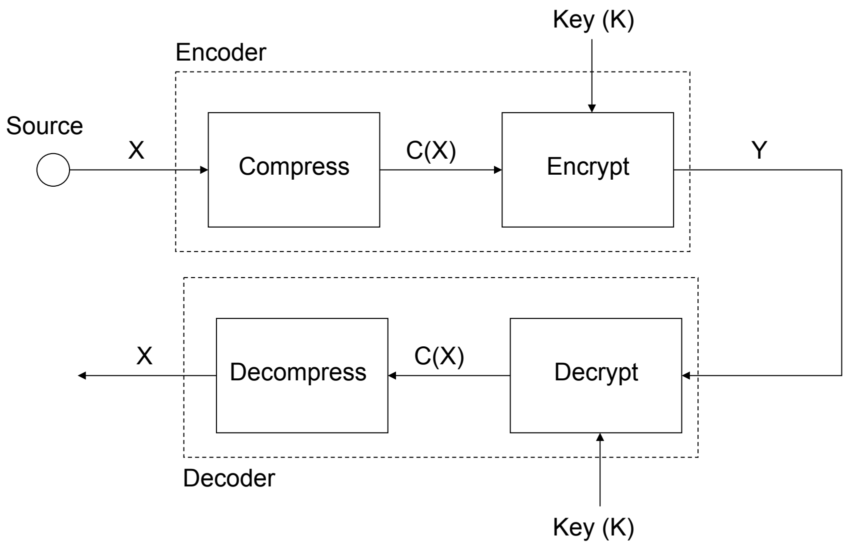

Figure 49: Traditional system with compression preceding encryption.

We are interested in systems that perform both compression and encryption, wherein the compressor has no access to the key. In such systems encryption is performed after

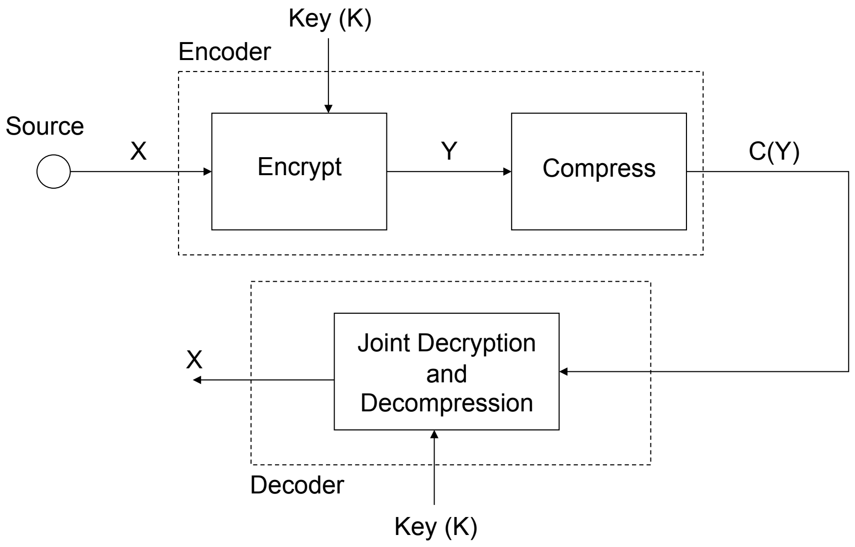

Figure 50: System with encryption subsequent to compression.

compression as depicted in Figure 49. This is a consequence of the traditional view which considers encrypted data hard to compress without knowledge of the key. In [31] a system similar to that of Figure 50 is considered instead, but with the order of encryption and compression reversed, while only the encryptor has access to the key. The authors consider encryption of a plaintext X using a one-time pad scheme with a finite-alphabet key (pad) K, to generate the ciphertext Y using

$$Y_i \triangleq X_i \oplus K_i.$$

This is followed by compression, which is agnostic of K, to generate the compressed ciphertext $C(Y)$.

The key insight underlying the approach in [31] is that the problem of compression in this case can be formulated as a Slepian-Wolf coding problem. In this formulation the ciphertext Y is cast as a source, and the shared key K is cast as the decoder-only side-information. The joint distribution of the source and side-information can be determined from the statistics of the source. For example, in the binary case with a uniformly distributed K_i and X_i with $\Pr[X_i = 1] = p$, we have

$$\Pr(Y_i \neq k | K_i = k) = p. \tag{87}$$

The decoder has knowledge of K and the source statistics. It uses this knowledge to reconstruct the ciphertext Y from the compressed message $C(Y)$, and to subsequently decrypt the plaintext X. This formulation is leveraged in [31] to show that exactly the same lossless compression rate, $H(X)$, can be achieved asymptotically by the system shown in Figure 50, as by the one in Figure 49, and all the while maintaining information-theoretic security.

The one-time pad and stream ciphers, while convenient for analysis, are not the only forms of encryption in practice. In fact, the prevalent method of encryption uses block ciphers, thus an obviously desirable extension of the technique in [31] would be to conventional encryption schemes such as AES. Attempting to do so, however, proves to be problematic. The method in [31] leverages the fact that in a one-time pad encryption scheme there exists a simple symbol-wise correlation between the key K and the ciphertext Y, as seen in Eq. (87). For block ciphers such as AES no such correlation is known. Any change in the plaintext is diffused in the ciphertext and quantifying the correlation (or the joint probability distribution) of the key and the ciphertext is believed to be computationally infeasible.

In the remainder of this chapter, we show to how circumvent this problem by exploiting the chaining modes popularly used with block ciphers and present methods for compressing data encrypted with block ciphers without knowledge of the key. As in [31], the proposed methods are based on the use of Slepian-Wolf coding.

To begin we define the notion of a *post-encryption compression* (PEC) scheme which is used throughout this chapter:

Definition 5.1. *Let $\mathcal{E} = (Gen, Enc, Dec)$ be an encryption scheme with plaintext domain $\mathcal{X}$ and ciphertext range $\mathcal{Q}$, and let $\mathcal{P}$ be a probability distribution over $\mathcal{X}$. Let C be a compression function defined over $\mathcal{Q}$, and D a (possibly probabilistic) decoding function with the property that for any key K generated by Gen, $D_K(C(Enc_K(X)) = X$ with probability $1-\delta$. Then the pair (C, D) is said to be $(\mathcal{E}, \mathcal{P}, \delta)$-$\mathsf{PEC}$ scheme.*

Note that in this definition D is given access to the encryption key while C operates independently of the key. Both C and D may be built for a specific plaintext probability distribution $\mathcal{P}$, for only in such case may correct decoding be guaranteed with high probability. We often assume that $\mathcal{P}$ is efficiently samplable, which means that there exists an efficient randomized algorithm whose output distribution is $\mathcal{P}$. The probability of error δ is taken over the choice of X and the choice of random coins if D is randomized. To simplify notation, we shall omit the $(\mathcal{E}, \mathcal{P}, \delta)$ designation if evident from the context, and use the term PEC scheme.

PEC schemes may be tailored to a specific cipher, but most often, one is interested in schemes that can simultaneously support different ciphers, such as AES and DES, or even an entire family of encryption schemes. Since all PEC schemes presented in this chapter work with any block cipher, we say that they are *generic*. A generic PEC scheme cannot be tailored to the specific details of particular ciphers but rather handles ciphers as black boxes. That is, the decoder D does not need to know the specifics of the encryption scheme nor its encryption/decryption key. It suffices that D has access to a pair of encryption and decryption oracles, denoted by *Enc* and *Dec*, that provide D with encryptions and decryptions, respectively, for any plaintext or ciphertext queried by D.

5.2 Compressing Block-cipher Encryption

As opposed to stream ciphers, such as the one-time pad, block ciphers are highly nonlinear and the correlation between the key and the ciphertext is, by design, hard to characterize. If a block cipher operates on each block of data individually identical inputs will produce identical outputs. While this weakness does not necessarily enable an unauthorized user to decipher an individual block, it can reveal valuable information; for example, about frequently occurring data patterns. To address this problem various chaining modes, also called modes of operation, are used in conjunction with block ciphers. The idea is to randomize each plaintext block by using a randomization vector derived from previous encryptor inputs or outputs. This randomization prevents identical plaintext blocks from being encrypted into identical ciphertext blocks.

Consider a sequence of plaintext blocks $\mathbf{X}^n = \{X_i\}_{i=1}^n$, where each block X_i is drawn from the set $\mathcal{X}^m = \{0,1\}^m$. Assume that the blocks X_i are generated by an i.i.d. source with a probability distribution P_X and encrypted with a block-cipher based private-key encryption scheme (Gen, Enc, Dec). In most cases of interest, block-cipher based encryption schemes use an initialization vector (IV) that is drawn uniformly at random from $\mathcal{X}^m$ by the encryption algorithm Enc_K. Let the encryption algorithm be characterized by the mapping $Enc_K : (\mathcal{X}^m)^n \to \mathcal{X}^m \times (\mathcal{X}^m)^n$. For the sequence of plaintext blocks $\mathbf{X}^n$ at input the encryption algorithm generates $Enc_K(\mathbf{X}^n) = \{\text{IV}, \mathbf{Y}^n\}$, where $\mathbf{Y}^n = \{Y_i\}_{i=1}^n$ denotes a sequence of ciphertext blocks Y_i in $\mathcal{X}^m$. The problem at hand is to compress $Enc_K(\mathbf{X}^n)$ without knowledge of K.

The remainder of this section is devoted to a description of compression schemes for various chaining modes used in conjunction with block ciphers. Initially, we turn attention to the cipher block chaining (CBC) mode, which is the most common mode of operation used with block ciphers (such as in Internet protocols TLS and IPsec). Afterwards, we discuss the output feedback (OFB) mode and the cipher feedback (CFB) mode, where compression is a relatively straightforward extension of the methods described in [31]. Lastly, we consider the electronic code book (ECB) mode, where our treatment provides fundamental insight into the feasibility of compression of data compressed with block ciphers without chaining.

5.2.1 Cipher Block Chaining

The most common mode of operation is CBC. Depicted in Figure 51, block ciphers in CBC mode are employed as the default mechanism in widespread security standards such as IPSec [34] and TLS/SSL [15] and hence it is a common method of encrypting internet traffic.

In the CBC mode each plaintext block X_i is randomized prior to encryption, by being XOR-ed with the ciphertext block Y_{i-1} corresponding to the previous plaintext block X_{i-1} to obtain $\tilde{X}_i$. The ciphertext block Y_i is generated by applying the block cipher with key K to the randomized plaintext block $\tilde{X}_i$ according to

$$Y_i = B_K\big(X_i \oplus Y_{i-1}\big), \tag{88}$$

where $Y_0 = \text{IV}$ and $B_K : \mathcal{X}^m \to \mathcal{X}^m$ is the block cipher mapping using the key K. At the

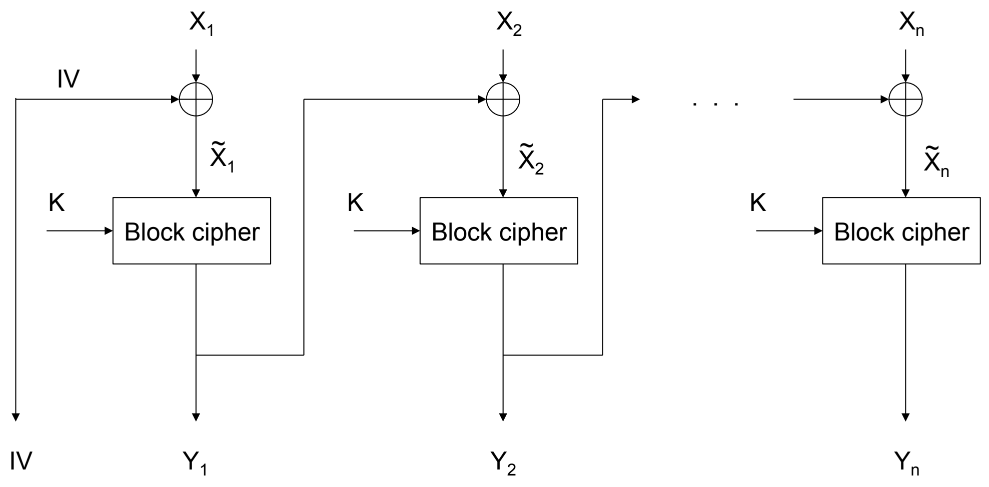

Figure 51: Cipher block chaining.

output of the encryption algorithm we have $Enc_K(\mathbf{X}^n) = (\text{IV}, \mathbf{Y}^n)$.

Notice that, contrary to OFB and CFB modes, ciphertext blocks in Y_i are not obtained by a bitwise XOR operation, but rather as outputs of highly nonlinear block ciphers; therefore, the compression methods from [31] cannot be applied directly in CBC mode.

The key insight underlying the proposed method for compression can now be described. While the statistical relationship between the key K and the ciphertext block Y_i is hard to characterize, the joint probability distribution of the randomization vector Y_{i-1} and the block cipher input $\tilde{X}_i$ is easy to characterize, as it is governed by the probability distribution P_X of the plaintext block X_i. For example, in the i.i.d source case considered here, Y_{i-1} and $\tilde{X}_i$ are related through a symbol-wise model governed by P_X. This correlation induced by the chaining mode can be exploited to allow compression of encrypted data using Slepian-Wolf coding, as we will now show.

Let $(C_{\text{CBC}}, D_{\text{CBC}})$ be a Slepian-Wolf code with encoding rate R and block length m. Denote the Slepian-Wolf encoding function as $C_{\text{CBC}} : \mathcal{X}^m \to \{1, \ldots, 2^{mR}\}$, and the Slepian-Wolf decoding function as $D_{\text{CBC}} : \{1, \ldots, 2^{mR}\} \times \mathcal{X}^m \to \mathcal{X}^m$. The proposed compression method is illustrated in Figure 52. The input to the compressor is $Enc_K(\mathbf{X}^n) = (\text{IV}, \mathbf{Y}^n)$. Since Y_i is in $\mathcal{X}^m$, the length of the input sequence in bits is $(n+1)m \log |\mathcal{X}|$. The compressor applies the encoding function C_{CBC} to IV and each of the first $n-1$ ciphertext blocks independently, while the n-th block is left unchanged. Thus, the output of the compressor is

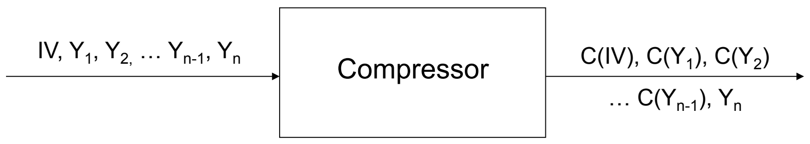

Figure 52: Compressor.

the sequence $(C_{\text{CBC}}(\text{IV}), C_{\text{CBC}}(Y_1), \dots C_{\text{CBC}}(Y_{n-1}), Y_n)$. The length of the output sequence is $nmR + m\log|\mathcal{X}|$ bits and the compressor factor

$$\frac{(n+1)\log|\mathcal{X}|}{nR + \log|\mathcal{X}|}$$

is achieved, which tends to $\log|\mathcal{X}|/R$ for large n. Note that the compressor does not need to know the key K and that this approach yields a compressed IV, which by itself (when there is no chaining) is incompressible; therefore, no performance loss is inflicted by the uncompressed last block.

A diagram of the proposed joint decompression and decryption method is shown in Figure 53. The received compressed sequence is decrypted and decompressed serially, from right to left. In the first step Y_n, which is received uncompressed, is decrypted using the key K to generate $\tilde{X}_n$. Next, Slepian-Wolf decoding is performed to reconstruct Y_{n-1} using $\tilde{X}_n$ as side-information, and the compressed bits $C_{\text{CBC}}(Y_{n-1})$. The decoder computes $\hat{Y} \triangleq D_{\text{CBC}}(C_{\text{CBC}}(Y_{n-1}), \tilde{X}_n)$, such that $\hat{Y} = Y_{n-1}$ with high-probability if the rate R is high enough. Once Y_{n-1} has been recovered by the Slepian-Wolf decoder, the plaintext block can now be reconstructed as $X_n = Y_{n-1} \oplus \tilde{X}_n$. The decoding process now proceeds serially, with Y_{n-1} decrypted to generate $\tilde{X}_{n-1}$, which then acts as the new side-information; this continues until all plaintext blocks are reconstructed.

For large m, it follows from the Slepian-Wolf theorem that the rate R required to ensure correct reconstruction of the $(i-1)$-th block with high probability is given by

$$\begin{aligned} R &= \frac{1}{m}H\big(Y_{i-1}|\tilde{X}_i\big) = \frac{1}{m}H\big(Y_{i-1}|Y_{i-1}\oplus X_i\big) \\ &= \frac{1}{m}H\big(Y_{i-1}, Y_{i-1}\oplus X_i|Y_{i-1}\oplus X_i\big) = \frac{1}{m}H\big(Y_{i-1}, X_i|Y_{i-1}\oplus X_i\big) \\ &= \frac{1}{m}H\big(X_i|Y_{i-1}\oplus X_i\big) \le \frac{1}{m}H(X_i). \end{aligned} \tag{89}$$

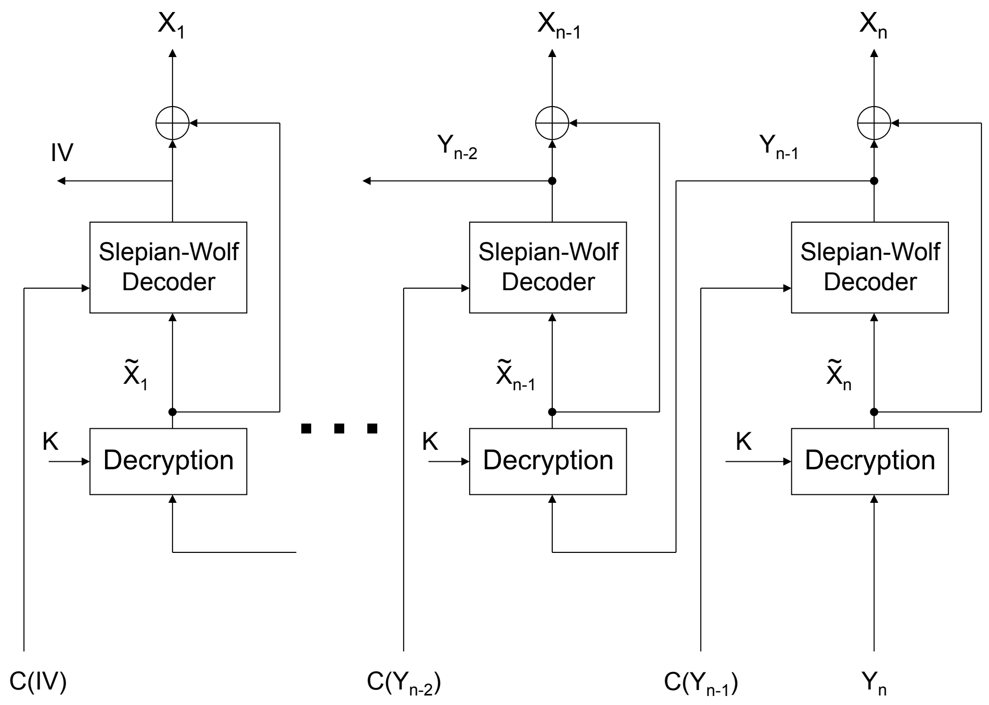

Figure 53: Joint decryption and decoding at the receiver. It is performed serially from right to left.

As IV is drawn uniformly at random, Y_i is uniformly distributed for all i. Consequently, the inequality $R \leq \frac{1}{m}H(X_i)$ in Eq. (89) becomes equality $R = \frac{1}{m}H(X_i)$.

In practice, as will be seen later, the block cipher input length m is typically small. In this case, the rate R is a function of P_X, the block cipher input length m, the acceptable decoding error probability, and the non-ideal Slepian-Wolf codes used.

5.2.2 Output Feedback

The setup depicted in Figure 54 is called output feedback (OFB) mode. Plaintext blocks in X_i are not directly encrypted with a block cipher. Rather, the block cipher is used to sequentially generate a sequence of pseudorandom blocks $\tilde{\mathbf{K}}^n = \{\tilde{K}_i\}_{i=1}^n$ which serve as a one-time pad to encrypt the plaintext blocks. At the output of the encryption algorithm we get $Enc_K(\mathbf{X}^n) = (\text{IV}, \mathbf{Y}^n)$.

Notice that each block $\tilde{K}_i$ is statistically independent of plaintext blocks; therefore, the OFB mode is analogous to the one-time pad encryption scheme and $\mathbf{Y}^n$ can be compressed in the same manner as in [31]. Most importantly, due to statistical independence between

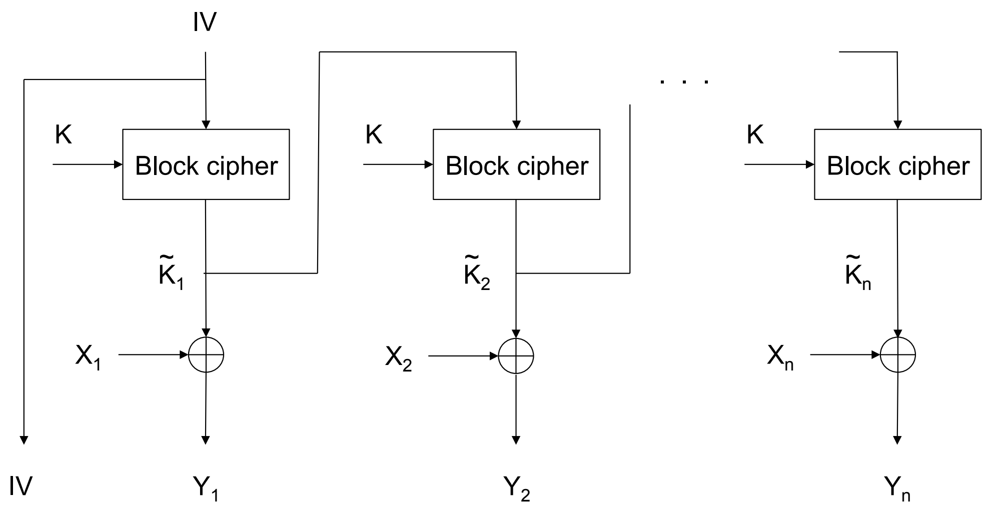

Figure 54: Output feedback.

$\tilde{\mathbf{K}}^n$ and $\mathbf{X}^n$ the block length of Slepian-Wolf codes that are used to compress $\mathbf{Y}^n$ can be chosen arbitrarily. The IV is uniformly distributed and therefore incompressible.

Formally, the compression and decompression algorithms can be described as follows. Let $(C_{\text{OFB}}, D_{\text{OFB}})$ be a Slepian-Wolf code with encoding rate R and block length nm. The Slepian-Wolf encoding function is denoted as $C_{\text{OFB}} : (\mathcal{X}^m)^n \rightarrow \{1, \ldots, 2^{mnR}\}$, and the Slepian-Wolf decoding function as $D_{\text{OFB}} : \{1, \ldots, 2^{mnR}\} \times (\mathcal{X}^m)^n \rightarrow (\mathcal{X}^m)^n$. Given an output sequence $Enc_K(\mathbf{X}^n) = (\text{IV}, \mathbf{Y}^n)$ from the encryptor, compression is achieved by applying the encoding function C_{OFB} to $\mathbf{Y}^n$, so that at the output we get $(\text{IV}, C_{\text{OFB}}(\mathbf{Y}^n))$. Note that the function C_{OFB} does not require knowledge of K.

Decompression and decryption are performed jointly. The receiver gets $(\text{IV}, C_{\text{OFB}}(\mathbf{Y}^n))$ and since it knows the secret key K, it generates the sequence of pseudorandom blocks $\tilde{\mathbf{K}}^n$ using IV. Subsequently, it applies the decoding function D_{OFB} to $(C_{\text{OFB}}(\mathbf{Y}^n), \tilde{\mathbf{K}}^n)$ and recovers $\mathbf{Y}^n$. The original sequence of plaintext blocks $\mathbf{X}^n$ then equals $\tilde{\mathbf{K}}^n \oplus \mathbf{Y}^n$.

By the Slepian-Wolf theorem, the compression rate approaches entropy of the source asymptotically in nm. Thus, even for finite m the rate still approaches entropy for large n.

5.2.3 Cipher Feedback

Next, we discuss the cipher feedback (CFB) mode depicted in Figure 55. Similarly to OFB, the plaintext blocks are not subjected to block cipher encryption and $\mathbf{Y}^n$ is obtained by XOR-ing plaintext blocks $\mathbf{X}^n$ with pseudorandom blocks $\tilde{\mathbf{K}}^n$. At the output of the encryption algorithm we get $Enc_K(\mathbf{X}^n) = (\text{IV}, \mathbf{Y}^n)$.

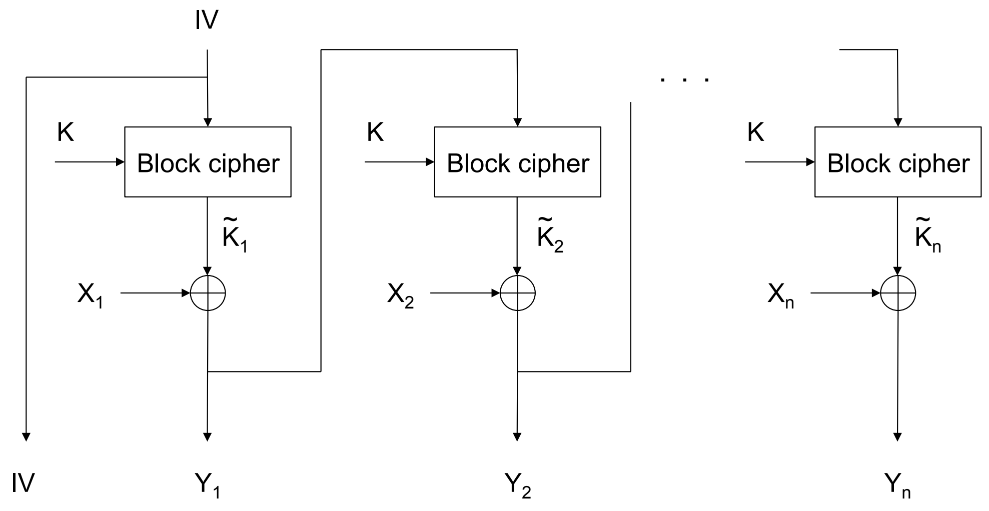

Figure 55: Cipher feedback.

However, in CFB mode the pseudorandom blocks $\tilde{\mathbf{K}}^n$ are not independent of $\mathbf{X}^n$ as they were in OFB. If the block cipher using a secret key K is characterized by the bijective mapping $B_K : \mathcal{X}^m \to \mathcal{X}^m$, then each block $\tilde{K}_i$ depends on the preceding plaintext block X_{i-1} according to

$$\tilde{K}_i = B_K(X_{i-1} \oplus \tilde{K}_{i-1}), \tag{90}$$

where X_0 equals IV.

Due to statistical dependence between $\tilde{\mathbf{K}}^n$ and $\mathbf{X}^n$, the proposed compression algorithm for CFB mode operates on individual ciphertext blocks Y_i, rather then on all blocks in $\mathbf{Y}^n$ at once as in OFB mode. Without this distinction, the joint decompression and decryption as proposed in [31] would not be possible. Let $(C_{\text{CFB}}, D_{\text{CFB}})$ be a Slepian-Wolf code with encoding rate R and block length m. The Slepian-Wolf encoding function is denoted as $C_{\text{CFB}} : \mathcal{X}^m \to \{1, \ldots, 2^{mR}\}$, and the Slepian-Wolf decoding function as $D_{\text{CFB}} : \{1, \ldots, 2^{mR}\} \times \mathcal{X}^m \to \mathcal{X}^m$. Then compression is achieved by applying the encoding function C_{CFB} to each of the ciphertext blocks in $\mathbf{Y}^n$ individually, giving a compressed representation of $Enc_K(\mathbf{X}^n)$ as $(\text{IV}, C_{\text{CFB}}(Y_1), \ldots, C_{\text{CFB}}(Y_n))$. Here again C_{CFB} does not require knowledge of K.

Joint decompression and decryption, depicted on Figure 56 is performed sequentially from left to right, since decryption of the ith ciphertext block requires knowledge of the $(i-1)$th plaintext block. Initially, IV is mapped to $\tilde{K}_1$ by the block cipher, then the decoder

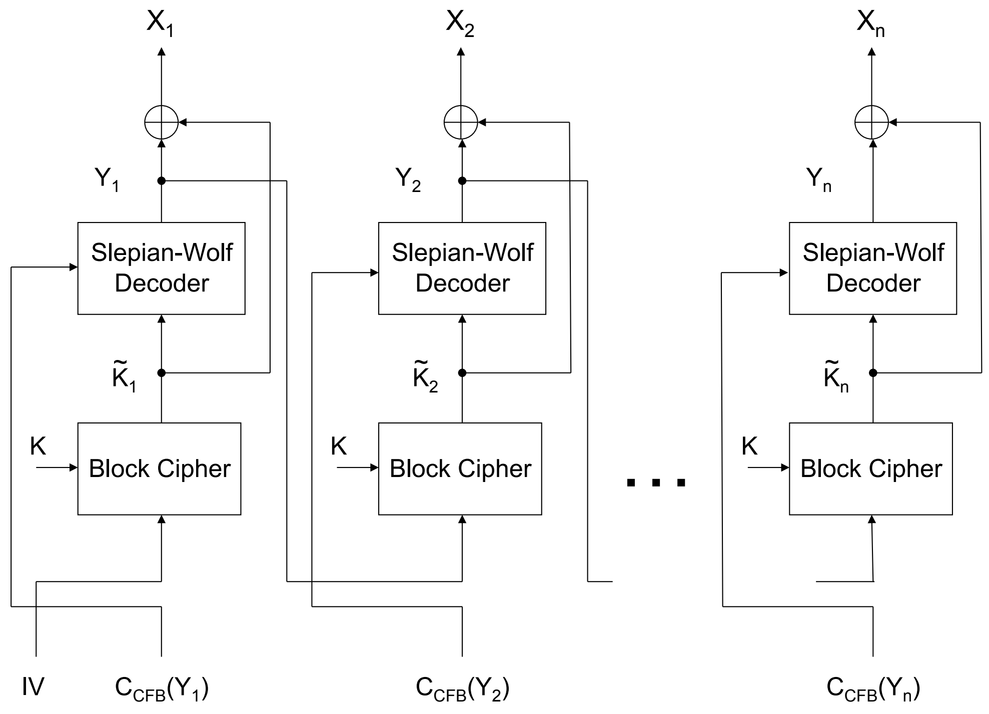

Figure 56: Joint decryption and decoding in CFB mode at the receiver is performed serially from left to right.

function D_{CFB} is applied to $(C_{\text{CFB}}(Y_1), \tilde{K}_1)$ to obtain Y_1, and now the first plaintext is obtained according to: $X_1 = \tilde{K}_1 \oplus Y_1$. Subsequently, Y_1 is mapped to $\tilde{K}_2$, the same process is repeated to obtain X_2, and so on until all remaining plaintext blocks $\{X_i\}_{i=3}^{n}$ are recovered.

In contrast to OFB mode, this compression scheme is generally optimal only if m is very large. For finite m, the compression is generally suboptimal, even for very large n.

5.2.4 Electronic Code Book

As we have seen, ciphertexts generated by a block cipher in OFB, CFB, and CBC modes can be compressed without knowledge of the encryption key. The compression schemes that we presented rely on the specifics of chaining operations and a natural question arises as to what extent can the output of a block cipher be compressed without chaining, i.e. when a block cipher is applied to a single block of plaintext. This mode of operation, depicted in Figure 57, is called the electronic code book (ECB) mode.

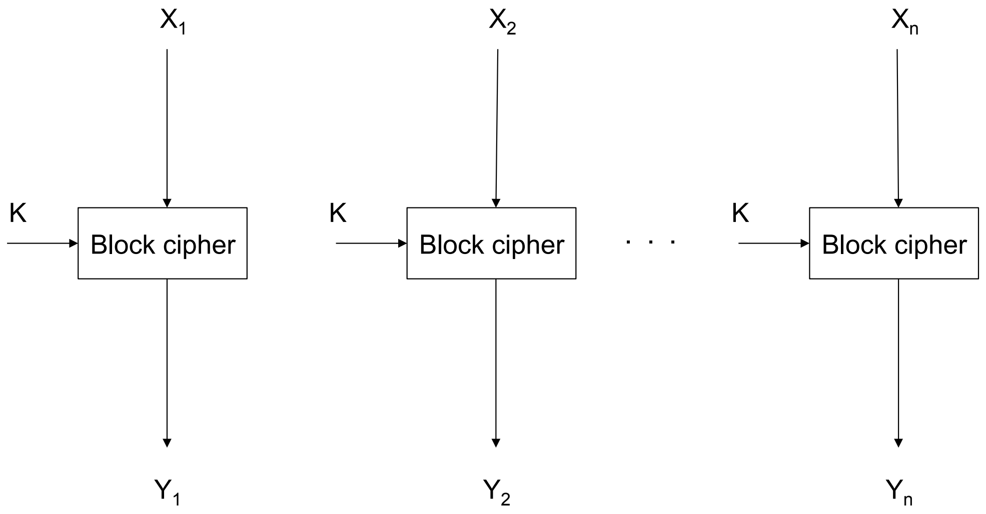

Figure 57: Electronic Codebook.

Since a block cipher with fixed key is a permutation, the entropy of a ciphertext is the same as the entropy of a plaintext. In effect, compression in ECB mode is theoretically possible, but the question is, whether it is possible to design a generic and efficient post-encryption compression scheme such as for other modes of operation.

We claim the answer to this question is negative, except for some low-entropy distributions or very low compression rates (e.g., compressing a ciphertext by only a few bits). For a given compressed ciphertext $C(Y_i)$ the decoder cannot do significantly better than to operate in one of the following two exhaustive decoding strategies:

(1) enumerate all possible plaintexts in decreasing order of probability and compute a ciphertext for each plaintext until the ciphertext Y_i that compresses to $C(Y_i)$ is found;

(2) enumerate all ciphertexts that compress to $C(Y_i)$ and decrypt them to find the original plaintext X_i.

We show that a generic compression scheme that compresses the output of a block cipher and departs significantly from one of the two strategies can be converted into an algorithm that breaks the security of the block cipher. In other words, a scheme that compresses the output of a secure block cipher either requires an infeasible amount of computation, i.e. as much as needed to break the block cipher, or it must follow one of the two exhaustive strategies. If (C, D) is a PEC scheme that follows any of the above exhaustive strategies, we say that (C, D) is an *exhaustive PEC scheme.*

The following result on the compressibility of block ciphers in ECB mode can now be stated.

Theorem 3. *Let B be a secure block cipher*[1]*, let $\mathcal{E} = (Gen, Enc, Dec)$ be an encryption scheme, where Gen chooses K uniformly from $\mathcal{K}$ and $Enc = B_K$, $Dec = B_K^{-1}$, and let $\mathcal{P}$ be an efficiently-samplable plaintext distribution. Let (C, D) be a $(\mathcal{E}, \mathcal{P}, \delta)$-PEC scheme. If (C, D) is generic for block ciphers, then (C, D) is either exhaustive or computationally infeasible (or both).*

The proof of Theorem 3 is given in Appendix A.2, which also includes more details on the computational bounds that were omitted in Theorem's statement for simplicity. The following remarks are worth noting:

(a) The exhaustive strategies are infeasible in most cases, but for very low-entropy plaintext distributions or for very low compression rates these strategies can be efficient. For example, consider a plaintext set $\mathcal{X}$ of 1,000 uniformly distributed 128-bit values. One can compress the output of a 128-bit block cipher applied to this set by truncating the ciphertext to 40 bits. Using a birthday-type bound it can be shown that the probability of two values in $\mathcal{X}$ being mapped to the same ciphertext is about 2^{-20}. Hence, the exhaustive strategy (1) would succeed in recovering the correct plaintext with probability $1 - 2^{-20}$. In general, the compression capabilities of this strategy will depend on the guessing entropy [53, 7] of the underlying plaintext distribution. Another compression method is to drop some bits of the ciphertext and let the decoder search for the original ciphertext until the correct plaintext is recovered, as in the strategy (2). This assumes that the dropping some bits allows for almost-unique

[1] *For the definition of a secure block cipher see Appendix A.1.*

decodability. The fact that these exhaustive compression strategies can be efficient for some plaintext distributions shows that Theorem 3 cannot not exclude existence of efficient generic coding schemes for some plaintext distributions.

(b) Ciphertext compression is theoretically possible. If efficiency is not of concern, one could consider a brute-force compression algorithm that first breaks the block cipher by finding its key[2], uses the key to decrypt, and then compresses the plaintext. If several blocks of plaintext are compressed sequentially, they can be re-encrypted by the compression algorithm resulting in an effective ciphertext compression. This method might be impractical, but it shows that the existence of generic compressors can be ruled out only if the PEC scheme is subject to computational constraints. Moreover, the above method could work efficiently against an insecure block cipher (in which case the key can found efficiently), and this confirms that both efficiency and security are essential to our result.

(c) Theorem 3 holds for generic compressors that do not use the internals of an encryption algorithm or the actual key in the (de)compression process. It does not rule out the existence of good PEC schemes for specific secure block ciphers, like for instance AES. To achieve compression, though, the PEC scheme would have to be contingent on the internal structure of a block cipher.

5.3 Compression Performance

Codes used to compress stream ciphers can have arbitrary block lengths, since the method itself does not directly impose any constraints on the block length. On the other hand, the compression methods proposed for CFB and CBC mode must operate block-wise, since decompression occurs serially and depends upon the preceeding decompressed block. In effect, the block length of Slepian-Wolf codes must be equal to the block cipher input length m.

[2]This can be done by exhaustive search based on a known plaintext-ciphertext pair or (for suitable plaintext distributions) by decrypting a sequence of encrypted blocks and finding a key which decrypts all blocks into elements of the underlying probability distribution.

Efficiency of the Slepian-Wolf compression depends on the performance of underlying Slepian-Wolf codes and it was shown in [27, 28] that it approaches entropy with $O(\sqrt{\frac{\log n}{n}})$, which is considerably slower than $O(\frac{1}{n})$ of arithmetic coding. It follows that for efficient Slepian-Wolf compression the block length of Slepian-Wolf codes must be long.

Slepian-Wolf codes over finite block lengths have non-zero FERs, which implies that the receiver will occasionally fail to recover $E_K(X_{i-1})$ correctly. Such errors must be dealt with on the system level, as they can result in catastrophic consequences due to error-propagation to subsequent blocks. In the following it is assumed that as long as FER is low enough, the system can recover efficiently, for instance by supplying the uncompressed version of the erroneous block to the receiver. The compression performance will therefore depend on the target FER.

We consider a binary i.i.d. source with a probability distribution $\Pr(X = 1) = p$ and $\Pr(X = 0) = 1 - p$, which produces plaintext bits. A sequence of plaintext bits is divided into blocks of size m, which are encrypted and compressed as described in Section 5.2. The receiver's task is to reconstruct $E_K(X_{i-1})$ from $\tilde{X_i}$ and side information $C(E_K(X_{i-1}))$. For the considered source Slepian-Wolf decoding is equivalent to error-correction over a binary symmetric channel (BSC) and thus the underlying codes should yield a FER that is lower or at most equal to the target FER over the BSC. Compression efficiency can be evaluated using two methods:

(a) fix probability p and determine the compression rate of a Slepian-Wolf code that satisfies the target FER; or

(b) pick a well-performing Slepian-Wolf code and determine the maximum probability p for which target FER is satisfied.

We use LDPC codes and evaluate compression performance according to method (b). In our simulations we use two LDPC codes, of which the first yields compression rate 0.5 and has degree distribution pair $\lambda(x) = 0.3317x + 0.2376x^2 + 0.4307x^5$, $\rho(x) = 0.6535x^5 + 0.3465x^6$ and the second yields compression rate 0.75 and degree distribution pair $\lambda(x) = 0.4249x + 0.0311x^2 + 0.5440x^4$, $\rho(x) = 0.8187x^3 + 0.1813x^4$. Both codes are constructed with the PEG algorithm. Belief propagation is used for decoding and the maximum number of iterations is set to 100.

Table 14: Attainable compression rates for block length $m = 128$ bits.

Target FER	Compression Rate	p	Source Entropy
10^{-3}	0.50	0.026	0.1739
10^{-4}	0.50	0.018	0.1301
10^{-3}	0.75	0.068	0.3584
10^{-4}	0.75	0.054	0.3032

Simulation results for block lengths of 128 and 1024 bits are shown in Tables 14 and 15, respectively. First, consider the current specification of the AES standard [52], where m is 128 bits. At FER of 10^{-3} the binary source to be compressed to rate 0.5, can have the probability p of at most 0.026, where its entropy equals 0.1739. The large gap between the achievable compression rate and the entropy is a consequence of a very short block length. Note that for a fixed compression rate the maximum probability p decreases with the decreasing target FER.

Table 15: Attainable compression rates for $m = 1024$ bits.

Target FER	Compression Rate	p	Source Entropy
10^{-3}	0.50	0.058	0.3195
10^{-4}	0.50	0.048	0.2778
10^{-3}	0.75	0.134	0.5710
10^{-4}	0.75	0.126	0.5464

An increase in block length to 1024 bits results in a considerable improvement in performance (see Table 15). For instance, at FER $= 10^{-3}$ and compression rate 0.5, the source can now have probability p up to 0.058. In future block cipher designs one could consider longer block sizes, in order to allow for better post-encryption compression.

5.4 Concluding Remarks

We considered compression of data encrypted with block ciphers without knowledge of the key. Contrary to the widespread belief that such data are practically incompressible, we show that compression can be attained. Our method is based on the Slepian-Wolf coding and hinges on the fact that chaining modes widely used with block ciphers introduce a simple symbol-wise correlation between successive blocks of data. Further, we showed there exists a fundamental limitation to compressibility of data encrypted with block ciphers when no chaining mode is employed.

Some simulation results are presented for binary memoryless sources. The results, while still far from theoretical limits, indicate that considerable compression gains are practically attainable with block ciphers, and improved performance can be expected with future increase of block sizes.

APPENDIX A

SECURITY OF THE CBC COMPRESSION SCHEME AND COMPRESSIBILITY OF BLOCK CIPHERS IN ECB MODE

In this appendix chapter we use standard techniques in the cryptographic literature to prove that the compression schemes proposed in Chapter 5 do not compromise the security of the original encryption scheme. We also show that there exists a fundamental limitation to the compression capability when the input to the block cipher is applied to a single-block message without chaining as in ECB mode (see Section 5.2.4).

A.1 Some Definitions from Cryptography

For completeness, we recall some standard definitions from cryptography that are used in this paper. See [33] for details. These definitions assume a fixed computational model over which time complexity is defined.

Definition A.1 (Block Cipher). *A block cipher B with block size m is a keyed family $\{B_K\}_{K \in \mathcal{K}}$ where for each K, B_K is a permutation over m bits*[1] *and $\mathcal{K}$ is the set of all possible keys.*

The security of a block cipher is defined via the notion of *indistinguishability.* Ideally, we would like the behavior of a block cipher to be indistinguishable by computational means from that of a purely random permutation over m bits. However, since a block cipher is a much smaller family of permutations than the family of *all* permutations, the above is not fully achievable. Yet, if we restrict our attention to "computationally feasible distinguishers" then we can obtain a meaningful notion of security applicable to actual block ciphers such as AES.

Hence, the main ingredient in such definition is that of a *distinguisher.* A distinguisher *Dist* is defined as a randomized algorithm with oracle access to two m-bit permutations

[1] *By a* permutation over m bits *we mean a deterministic bijective function over* $\{0,1\}^m$.

Enc and Dec where $Dec = Enc^{-1}$. $Dist$ can make arbitrary queries to the oracles and eventually outputs a single bit 0 or 1. We consider the runs of $Dist$ in two cases: When the oracles are instantiated with a truly random permutation and when instantiated with a block cipher (i.e., with the functions B_K and B_K^{-1} where K is chosen with uniform probability from the set $\mathcal{K}$). Let P_{REAL} be the probability that $Dist$ outputs a 1 when the oracle was instantiated with B, where P_{REAL} is computed over all random coins of $Dist$ and all choices of the key for B. Further, let P_{RP} be the probability that $Dist$ outputs a 1 when the oracle was instantiated with a random permutation, where P_{RP} is computed over all random coins of $Dist$ and all permutations Enc. Intuitively, we can think of $Dist$ as trying to decide if the oracles are instantiated with a random permutation or with a block cipher; hence, a distinguisher is considered successful if the difference $|P_{REAL} - P_{RP}|$, which is referred to as the advantage, is non-negligible. Formally, this leads to the following definition.[2]

Definition A.2 (Block Cipher Security)**.** *A block cipher B is called (T, ε)-secure if no distinguisher $Dist$ that runs in time T has advantage larger than ε.*

This definition tries to capture the idea that, for secure block ciphers, even distinguishers that have the ability to run for extremely large time T, say $T = 2^{80}$, gain only negligible distinguishing advantage, say $\varepsilon = 2^{-40}$. In other words, for any practical purpose the quality of the block cipher is as good as if it was instantiated by a "perfect cipher" (purely random permutation).

A secure block cipher by itself does not constitute a secure private-key encryption scheme due to its deterministic nature. Namely, two identical plaintexts are mapped into two identical ciphertexts, therefore valuable information about data patterns can be leaked to eavesdroppers. Rather, secure block ciphers are used as building blocks that can be used to construct private-key encryption schemes that eliminate this vulnerability. Note that a secure private-key encryption scheme must be probabilistic or stateful.

Toward a formal definition of a secure encryption scheme, consider the following experiment:

[2]This definition corresponds to the notion of strong pseudorandom permutation [3].

Definition A.3 (CPA Indistinguishability Experiment)**.** *Consider an adversary $\mathcal{A}$ and an encryption scheme (Gen, Enc, Dec). The* chosen plaintext attack (CPA) indistinguishability experiment $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{cpa}}$ *is defined as follows:*

1. *a key k is generated by running Gen;*
2. *the adversary $\mathcal{A}$ has oracle access to $Enc_K(\cdot)$, and queries it with a pair of plaintexts X_0, X_1 of the same length;*
3. *the oracle randomly chooses a bit $b \leftarrow \{0,1\}$ and returns the ciphertext $q \leftarrow Enc_K(X_b)$, called the challenge, to $\mathcal{A}$;*
4. *the adversary $\mathcal{A}$ continues to have oracle access to $Enc_K(\cdot)$ and is allowed to make arbitrary queries. Ultimately it makes a guess about the value of b by outputting b';*
5. *the output of the experiment, $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{cpa}}$, is defined to be 1 if $b' = b$, and 0 otherwise. If $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{cpa}} = 1$, we say that $\mathcal{A}$ succeeded.*

Definition A.4 (Private-Key Encryption Security)**.** *A private-key encryption scheme (Gen, Enc, Dec) is called (T, ε)-*indistinguishable under chosen plaintext attacks *(or CPA-secure) if for every adversary $\mathcal{A}$ that runs in time T,*

$$\Pr\left[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{cpa}} = 1\right] < \frac{1}{2} + \varepsilon,$$

where the probability is taken over all random coins used by $\mathcal{A}$, as well as all random coins used in the experiment.

A.2 Proof of Theorem 3

The proof of Theorem 3 is by contradiction. We assume a generic PEC scheme (C, D) that departs in some noticeable way (later made more precise by means of the parameter ε) from the exhaustive strategies when it is applied to a block cipher B. Subsequently, we show how to use such a PEC scheme to build a distinguisher *Dist* that distinguishes with advantage strictly larger than ε between the block cipher B and a random permutation, in contradiction to the security of B.

For simplicity, and without loss of generality, we assume that D does not make redundant queries to *Enc* or *Dec*. Namely, no query X to *Enc* or query Y to *Dec* is repeated in a run. If X was output by *Dec* (resp., Y output by *Enc*) it is not entered into *Enc* (resp., into *Dec*). In addition, if the decoded output from D is X, we assume that X was either input to *Enc* or output by *Dec*. If D does not follow these rules, it can be modified to do so.

For $X \in_R \mathcal{P}$ [3] and $Y = Enc(X)$, assume that $C(Y)$ is passed as input to D. We say that D decodes correctly if it queries either X from *Enc* or Y from *Dec* during its run. In other words, D is not required to have the ability to identify the correct plaintext, which simplifies our presentation without weakening our results. On the contrary, it shows that even if such a relaxed decoding requirement is acceptable, the lower bound we prove still holds.

Finally, note that the formulation of the theorem assumes that the plaintext distribution $\mathcal{P}$ is efficiently samplable. This assumption is used in an essential way in our proof, though the efficiency requirement from the $\mathcal{P}$ sampler is very weak. In addition, we assume that, for a given compressed ciphertext $C(Y)$, one can sample uniformly from the set $\mathcal{Y}_{C(Y)} = C^{-1}(C(Y))$, which is the set of all ciphertexts mapped by C to $C(Y)$. Additional discussion related to this assumption is given after the proof.

The proof of Theorem 3 uses the following lemma.

Lemma A.1. *Let T be a time-bound parameter, let B be a (T, ε)-secure block cipher and let $\mathcal{E} = (Gen, Enc, Dec)$ be an encryption scheme, where Gen chooses $K \in_R \mathcal{K}$ and $Enc = B_K$, $Dec = B_K^{-1}$. Let $\mathcal{P}$ be a plaintext distribution samplable in time $T/4$ and let (C, D) be a generic $(\mathcal{E}, \mathcal{P}, \delta)$-PEC scheme. Then either (C, D) runs in time that exceeds $T/4$ (and hence is infeasible[4]) or the following holds:*

(i) Let $X \in_R \mathcal{P}$ and $Y = Enc(X)$. Consider a run of D on input $C(Y)$ in which D queries $Enc(X)$, and let X' be a random element drawn from $\mathcal{P}$ independently of X. Then, the probability that D queries $Enc(X')$ before $Enc(X)$ is at least $1/2 - \varepsilon/(1 - \delta)$.

[3] We use $X \in_R \mathcal{P}$ to denote that X is chosen at random according to the probability distribution $\mathcal{P}$.

[4] *For secure block ciphers, a distinguisher should not be able to attain more then a negligible advantage even if it runs for extremely large time T, say $T = 2^{80}$ (see Appendix A.1). Thus, a PEC scheme that runs in time $T/4$ would be considered infeasible.*

(ii) Let $X \in_R \mathcal{P}$ and $Y = Enc(X)$. Consider a run of D on input $C(Y)$ in which D queries $Dec(Y)$, and let Y' be an element drawn uniformly from $\mathcal{Y}_{C(Y)} = C^{-1}(C(Y))$. Then, the probability that D queries $Dec(Y')$ before $Dec(Y)$ is at least $1/2 - \varepsilon/(1-\delta)$.

We first show how the Lemma A.1 suffices to prove Theorem 3.

Proof of Theorem 3. For case (i), the probability that X is queried first is within $\varepsilon/(1-\delta)$ of the probability that X' is queried first, where the latter is the probability of querying a plaintext that bears no information (the run of D is independent of X'). Assuming that $\delta < 1/2$, we get $\varepsilon/(1-\delta) < 2\varepsilon$ and since ε is negligible (say 2^{-40}) so is 2ε. This implies a plaintext-exhaustive strategy by D. Similarly, for case (ii), the probability that the value Y is computed first is within $\varepsilon/(1-\delta)$ of the probability that an independent $Y' \in_R \mathcal{Y}_{C(Y)}$ is computed first. This implies a ciphertext-exhaustive strategy by D. $\square$

Proof of Lemma A.1. First, consider the error-free case, i.e. $\delta = 0$. We show that if (C, D) runs in time less than $T/4$ and conditions (i), (ii) do not hold, we can build a distinguisher (see Appendix A.1) that runs in time at most T and distinguishes between B and a random permutation with an advantage larger than ε, in contradiction to the security of B. A distinguisher interacts with oracles *Enc* and *Dec* and its goal is to identify whether the oracles are instantiated with the real block cipher B or with a random permutation. For clarity, we represent the output 0 from *Dist* by the symbol RP (*Dist* decided that the oracles are instantiated with a random permutation) and the output 1 by REAL (*Dist* decided that the oracles are instantiated by the block cipher B).

We build a distinguisher *Dist* that uses the scheme (C, D) and responds to the encryption and decryption queries made by D with its *Enc*/*Dec* oracles. When *Enc* is a random permutation, (C, D) may run much longer than when *Enc* is B. Thus, to bound the time complexity of *Dist*, we set a time limit $T' = T/4$, such that if the total time of (C, D) exceeds T', *Dist* stops as well. As we show below, the parameter T' is chosen as $T/4$ to ensure that the total running time of *Dist* is no more than T.

Initially, $Dist$ chooses $X \in_R \mathcal{P}$ and receives the value $Y = Enc(X)$ from its Enc oracle. It computes $C(Y)$ and passes it as input to D. In addition, $Dist$ chooses an independent $X' \in_R \mathcal{P}$ and independent $Y' \in_R \mathcal{Y}_{C(Y)}$. It is assumed that Y' can be sampled within time $T/4$. Subsequently, $Dist$ monitors the queries to Enc/Dec as requested by D and reacts to the following events:

1. if X is queried from Enc, stop and output REAL;
2. if X' is queried from Enc, stop and output RP;
3. if Y is queried from Dec, stop and output REAL;
4. if Y' is queried from Dec, stop and output RP.
5. if the run of (C, D) exceeds T', stop and output RP

It is possible that multiple such events take place in one run of D, for instance both X and X' may be queried from Enc. In such case $Dist$ stops as soon as it identifies first such event.

Let P_{REAL} and P_{RP} be defined as in Appendix A.1. We evaluate the advantage of $Dist$, namely, the difference $|P_{REAL} - P_{RP}|$. First, consider a run of $Dist$ when Enc/Dec are instantiated by a random permutation. The behavior of (C, D) depends on Y which is chosen at random and independently of X and X', therefore the run is independent of both X and X'. In effect, the probability that X is queried before X' is exactly 1/2. Similarly, if Y is queried from Dec, the behavior of D depends on $C(Y)$, while both Y and Y' have the same probability to be the chosen as the preimage of $C(Y)$. Therefore, the probability that Y precedes Y' is exactly 1/2. It follows that $Dist$ outputs REAL with probability at most 1/2 (exactly 1/2 for the X and Y cases and with probability 0 if $Dist$ exceeds time T'), i.e., $P_{RP} \leq 1/2$.

Now, consider a run of $Dist$ when Enc/Dec are instantiated by a block cipher B and its inverse, respectively. Assume, for contradiction, that (C, D) stops before time T' and either (i) the probability that X' is queried before X is strictly less than $1/2 - \varepsilon$ or (ii) the probability that Y' is queried before Y is strictly less than $1/2 - \varepsilon$. It follows that the probability that *Dist* outputs RP is strictly smaller than $1/2-\varepsilon$, therefore $P_{REAL} > 1/2+\varepsilon$.

Thus, *Dist* distinguishes with advantage $|P_{REAL} - P_{RP}|$, which is strictly larger than ε. The running time of *Dist* is upper-bounded by T: it includes three samplings (of X, X' and Y'), each assumed to take at most $T/4$ time, and the work of (C, D) which *Dist* runs for total time $T/4$ at most. In all, we have built a distinguisher against B that runs time T and has advantage larger than ε in contradiction to the security of the block cipher B.

Assume now that $\delta > 0$. When a positive probability of error for D is allowed, it can occur that D stops before time T' and before *Dist* sees X, X', Y or Y'. To deal with this situation, we add a clause to the specification of *Dist* saying that if (C, D) stops before time T' and before seeing any of the values X, X', Y, Y', then *Dist* chooses a random bit b and outputs REAL if $b = 1$ and RP if $b = 0$. We slightly increased the probability that X' is queried before X (or Y' before Y) in the block cipher case, however it is still negligibly far from 1/2. The proof is now a straightforward extension of the case when $\delta = 0$. □

Remark. The proof of Lemma A.1 assumes that the set $\mathcal{Y}_{C(Y)}$ is samplable in time $T/4$. While this assumption is likely to hold, we note that it is enough to know the size of $\mathcal{Y}_{C(Y)}$. In such case, rather than sampling $\mathcal{Y}_{C(Y)}$, the events 3) and 4) in the proof are replaced with the following one: if the number of queries to *Dec* performed by D exceeds $|\mathcal{Y}_{C(Y)}|/2$ before Y is queried, stop and output RP.

We now sketch the proof of the theorem when none of the above conditions holds, namely when $\mathcal{Y}_{C(Y)}$ is of unknown size and not samplable in time $T/4$. Assume that (C, D) performs noticeably better than the exhaustive strategies when the *Enc*/*Dec* oracles are instantiated with the block cipher B. Then it must hold that (C, D) runs noticeably faster when the *Enc* and *Dec* oracles are instantiated with the block cipher B than with a random permutation, for the exhaustive strategies are optimal for a random permutation (as shown above). Using this (assumed) discrepancy between the runs of (C, D) over B and the runs of (C, D) over a random permutation, we can build a distinguisher against B in contradiction to the security of B. In the following, we formalize this discrepancy and outline the construction of the distinguisher, where some straightforward details are omitted.

For any plaintext X let $T_B(X)$ denote the runtime of (C, D) on input X when *Enc*/*Dec* oracles are instantiated with the block cipher B. Further, let $T_R(X)$ denote the runtime of (C, D) on input X when *Enc*/*Dec* oracles are instantiated with a random permutation. We assume that there is a known time bound T_0 such that $T_B(X) < T_0$ for all X (we relax this assumption below) and there exists a non-negligible ε such that $\Pr[T_B(X) < T_R(X)] \geq 1/2 + \varepsilon$. The probability is over all choices of X and all random coins of (C, D). Further, it is over all key choices for B for T_B and over all random permutations for T_R. We build a distinguisher *Dist* as follows:

1. Choose $X \in_R \mathcal{P}$ and run (C, D) using the input oracles. If time T_0 is exceeded, stop and output RP, otherwise proceed to Step 2.

2. Let T_1 denote the running time of (C, D) in step (1). Run (C, D) again on X (same X as in step 1)) but this time ignore the given *Enc*/*Dec* oracles. Instead, answer queries from (C, D) with a random permutation. If time T_1 is exceeded then output REAL, otherwise output RP.

We have the following:

- If the *Enc*/*Dec* oracles are instantiated with B, step (1) always completes and in step (2) REAL is output with the probability that equals $\Pr[T_R(X) > T_B(X)]$, which is at least $1/2 + \varepsilon$.

- If the *Enc*/*Dec* oracles are instantiated with a random permutation, REAL is output only if $T_R(X) \leq T_0$ and the runtime on X in step (2) exceeds the runtime on X in step (1). The probability of the latter is at most $1/2$ since both runs are over a random permutation.

Thus, *Dist* will output REAL with probability at least $1/2 + \varepsilon$ when the oracles are instantiated with B, while it will output REAL with probability at most $1/2$ when the oracles are instantiated with a random permutation. It follows that *Dist* is a (T, ε)-distinguisher for B where $T = 2T_0$ since in each of the steps (1) and (2) *Dist* runs time at most T_0.

Note that the requirement that $T_B(X) < T_0$ for all X can be relaxed such that the joint probability of $T_B(X) < T_0$ and $T_B(X) < T_R(X)$ is at least $1/2 + \varepsilon$.

A.3 Security of the CBC Compression Scheme

In this section we formally prove that compression and decompression operations that we introduced on top of the regular CBC mode do not compromise security of the original CBC encryption. The proof follows standard techniques in the cryptographic literature, showing that any efficient attack against secrecy of a PEC scheme can be transformed into an efficient attack against the original CBC encryption.

We start by formalizing the notion of security of a PEC scheme as a simple extension of the standard definition of chosen plaintext attack (CPA) security recalled in Appendix A.1. The essence of the extension is that in the PEC setting the adversary is given access to a combined oracle $(Enc_K + C)(\cdot)$ which first encrypts the plaintext and then compresses the resultant ciphertext.

Definition A.5 (CPA-PEC Indistinguishability Experiment). *Consider an encryption scheme* $(Gen, Enc,$
$Dec)$ *and a PEC scheme* (C, D). *The* CPA-PEC indistinguishability experiment $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{cpa-pec}}$ *is defined as follows:*

1. *a key K is generated by running Gen;*

2. *the adversary $\mathcal{A}$ has oracle access to $(Enc_K + C)(\cdot)$, and queries it with a pair of test plaintexts X_0, X_1 of the same length;*

3. *the oracle randomly chooses a bit $b \leftarrow \{0, 1\}$ and returns the compressed ciphertext $c \leftarrow (Enc_K + C)(X_b)$, called the challenge, to $\mathcal{A}$;*

4. *the adversary $\mathcal{A}$ continues to have oracle access to $(Enc_K + C)(\cdot)$ and is allowed to make arbitrary queries. Ultimately it makes a guess about the value of b by outputting b';*

5. *the output of the experiment,* $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{cpa-pec}}$*, is defined to be* 1 *if* $b' = b$*, and* 0 *otherwise. If* $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{cpa-pec}} = 1$*, we say that* $\mathcal{A}$ *succeeded.*

Definition A.6 (Post-Encryption Compression Security)**.** *A PEC scheme* (C, D) *is called* (T, ε)*-indistinguishable under chosen plaintext attacks (*CPA-PEC-secure*) if for every adversary* $\mathcal{A}$ *that runs in time* T*,*

$$\Pr\left[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{cpa-pec}} = 1\right] < \frac{1}{2} + \varepsilon,$$

where the probability is taken over all random coins used by $\mathcal{A}$*, as well as all random coins used in the experiment.*

We now formulate the security of our CBC compression scheme by the following theorem.

Theorem 4. *Let* $\mathcal{E} = (Gen, Enc, Dec)$ *be a CBC encryption scheme that is* (T, ε)*-indistinguishable under chosen plaintext attacks, let* $\mathcal{P}$ *be an efficiently-samplable plaintext distribution, and let* (C, D) *be a* $(\mathcal{E}, \mathcal{P}, \delta)$*-PEC scheme. Then* (C, D) *is* $(T/T_C, \varepsilon)$*-indistinguishable under chosen plaintext attacks, where* T_C *is an upper bound on the running time of* C*.*

Proof. Our proof employs a reduction to the security of $\mathcal{E}$. Specifically, we show that if there exists an adversary $\mathcal{A}_C$ that runs in time T/T_C and is able to distinguish between two compressed encryptions, then there exists an adversary $A_{\mathcal{E}}$ that runs in time T and compromises the security of $\mathcal{E}$. Note that the latter implies a break of security of the underlying block cipher using the well-known result by Bellare et al. [3].

Assume, for contradiction, the existence of such an adversary $\mathcal{A}_C$. We construct an adversary $\mathcal{A}_{\mathcal{E}}$ as follows. $\mathcal{A}_{\mathcal{E}}$ invokes $\mathcal{A}_C$ and emulates its oracle $(Enc_K + C)(\cdot)$. That is, for every query X made by $\mathcal{A}_C$, $\mathcal{A}_{\mathcal{E}}$ uses its oracle $Enc_K(\cdot)$ to compute $Enc_K(X)$. Subsequently, it applies the compression algorithm C on $Enc_K(X)$ and forwards $C(Enc_K(X))$ to $\mathcal{A}_C$. When $\mathcal{A}_C$ outputs the two test messages X_0, X_1, $\mathcal{A}_{\mathcal{E}}$ outputs these messages to its own oracle $Enc_K(\cdot)$. Let c^* denote the challenge ciphertext that its oracle returns. $\mathcal{A}_{\mathcal{E}}$ computes $c = C(c^*)$ and forwards c to $\mathcal{A}_C$. When $\mathcal{A}_C$ outputs a bit b', $\mathcal{A}_{\mathcal{E}}$ outputs the same value.

Given that $\mathcal{A}_C$ makes at most T/T_C queries and that the running time of C is upper-bounded by T_C, $\mathcal{A}_{\mathcal{E}}$'s running time is at most T. Let $\frac{1}{2} + \epsilon$ denote the probability that $\mathcal{A}_C$

distinguishes successfully in its game. Clearly, it holds that $\mathcal{A}_{\mathcal{E}}$ distinguishes successfully in its game with the same probability, which is in contradiction to the security of $\mathcal{E}$.

$\square$

REFERENCES

[1] Aaron, A. and Girod, B., "Compression with side information using turbo codes," in *IEEE Data Compression Conf.*, pp. 252–261, 2002.

[2] Barnault, L. and Declercq, D., "Fast decoding algorithm for ldpc codes over gf(2^q)," in *Proc. International Symposium on Information Theory (ISIT)*, 2003.

[3] Bellare, M., Desai, A., Jokipii, E., and Rogaway, P., "A concrete security treatment of symmetric encryption: Analysis of the DES modes of operation," in *IEEE Proc. of 38th Annual Symp. on Foundations of Computer Science*, 1997.

[4] Bennatan, A. and Burshtein, D., "Design and analysis of nonbinary ldpc codes for arbitrary discrete memoryless channels," *IEEE Transactions on Information Theory*, vol. 52, pp. 549–583, Feb 2006.

[5] Blake, I. F., Seroussi, G., and Smart, N. P., *Eliptic Curves in Cryptography.* Cambridge University Press, 2000.

[6] Bloch, M. and Barros, J., *Physical Layer Security: From Information Theory to Security Engineering.* Cambridge University Press, to be published.

[7] Cachin, C., *Entropy Measures and Unconditional Security.* PhD thesis, ETH Zürich, 1997.

[8] Chung, S., *On the Construction of Some Capacity-Approaching Coding Schemes.* PhD thesis, Massachusetts Institute of Technology, 2000.

[9] Chung, S., Jr, G. F., Richardson, T., and Urbanke, R., "On the design of low-density parity-check codes within 0.0045 db of the shannon limit," *IEEE Communications Letters*, vol. 5, pp. 58–60, Feb 2001.

[10] Chung, S., Richardson, T., and Urbanke, R., "Analysis of sum-product decoding of low-density parity-check codes using a gaussian approximation," *IEEE Transactions on Information Theory*, vol. 47, pp. 657–670, Feb 2001.

[11] Csiszar, I. and Korner, J., "Broadcast channels with confidential messages," *IEEE Transactions on Information Theory*, vol. 24, pp. 339–348, May 1978.

[12] Davey, M. and MacKay, D., "Low-density parity check codes over gf(q)," *IEEE Communications Letters*, vol. 2, pp. 165–167, Jun 1998.

[13] Declercq, D. and Fossorier, M., "Decoding algorithms for nonbinary ldpc codes over gf(q)," *IEEE Transactions on Communications*, vol. 55, pp. 633–643, Apr 2007.

[14] Di, C., Proietti, D., Telatar, I. E., Richardson, T. J., and Urbanke, R. L., "Finite-length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Transactions on Information Theory*, vol. 48, pp. 1570–1579, Jun 2002.

[15] Dierks, T. and Rescorla, E., "The TLS protocol – version 1.2," in *RFC 5246*, Aug. 2008.

[16] ElGamal, T., "A public-key cryptosystem and a signature scheme based on discrete logarithms," in *CRYPTO '84*, pp. 10–18, Springer-Verlag (LNCS 196), 1984.

[17] Gallager, R. G., *Low-density parity-check codes.* MIT Press, 1963.

[18] Garcia-Frias, J., "Compression of correlated binary sources using turbo codes," *IEEE Communications Letters*, vol. 5, pp. 417–419, Oct. 2001.

[19] Gentry, C., "How to compress Rabin ciphertexts and signatures (and more)," in *CRYPTO '04*, pp. 179–200, Springer-Verlag (LNCS 3152), 2004.

[20] Ha, J., Kim, J., and McLaughlin, S., "Rate-compatible puncturing of low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 50, pp. 2824–2836, Nov 2004.

[21] Ha, J., Klinc, D., Kwon, J., and Mclaughlin, S. W., "Layered BP decoding for rate-compatible punctured LDPC codes," *IEEE Communications Letters*, vol. 11, pp. 440–442, May 2007.

[22] Ha, J., Kim, J., Klinc, D., and McLaughlin, S. W., "Rate-compatible punctured low-density parity-check codes with short block lengths," *IEEE Transactions on Information Theory*, vol. 52, pp. 728–738, Feb 2006.

[23] Ha, J. and Klinc, D., "Low-density parity-check codes with rate adaptability," *Telecommunications Review*, pp. 823–836, Dec 2006.

[24] Hagenauer, J., "Rate-compatible punctured convolutional codes (RCPC codes)," *IEEE Transactions on Communications*, vol. 36, pp. 389–400, Apr 1988.

[25] Harrison, W. K. and Mclaughlin, S. W., "Physical-layer security: Combining error control coding and cryptography," in *Proc. IEEE International Conference on Communications*, (Dresden, Germany), 2009.

[26] Harrison, W. K. and Mclaughlin, S. W., "Tandem coding and cryptography on wiretap channels: EXIT chart analysis," in *Proc. International Symposium on Information Theory (ISIT)*, (Seoul, South Korea), 2009.

[27] He, D., Lastras-Montaño, L., and Yang, E., "A lower bound for variable rate Slepian-Wolf coding," in *IEEE Inter. Symp. on Info. Theory*, (Seattle, WA), July 2006.

[28] He, D., Lastras-Montaño, L., and Yang, E., "On the relationship between redundancy and decoding error in Slepian-Wolf coding," in *IEEE Information Theory Workshop*, (Chengdu, China), Oct. 2006.

[29] Hocevar, D. E., "A reduced complexity decoder architecture via layered decoding," in *SIPS*, pp. 107–112, 2004.

[30] Hu, X., Eleftheriou, E., and Arnold, D., "Regular and irregular progressive edge-growth Tanner graphs," *IEEE Transactions on Information Theory*, vol. 51, pp. 386–398, Jan 2005.

[31] Johnson, M., Ishwar, P., Prabhakaran, V., Schonberg, D., and Ramchandran, K., "On compressing encrypted data," *IEEE Trans. Signal Processing*, vol. 52, pp. 2992–3006, Oct. 2004.

[32] Johnson, M., Wagner, D., and Ramchandran, K., "On compressing encrypted data without the encryption key," in *Proc. of the Theory of Crypto. Conf.*, (Cambridge, MA), Feb. 2004.

[33] Katz, J. and Lindell, Y., *Introduction to Modern Cryptography.* Chapman & Hall/CRC, 2007.

[34] Kent, S. and Seo, K., "Security achitecture for the internet protocol," in *RFC 4301*, Dec. 2005.

[35] Kim, J., Ramamoorthy, A., and Mclaughlin, S., "The design of efficiently-encodable rate-compatible ldpc codes," *IEEE Transactions on Communications*, vol. 57, pp. 365–375, Feb 2009.

[36] Kim, S. H., *Analysis of quasi-cyclic low-density parity-check codes and protograph codes.* PhD thesis, Seoul National University, Seoul, Korea, 2006.

[37] Klinc, D., Ha, J., Kim, J., and McLaughlin, S. W., "Rate-compatible punctured low-density parity-check codes for ultra wide band systems," in *Proc. Global Telecommunications Conference (GLOBECOM)*, (St. Louis, MO), Nov 2005.

[38] Klinc, D., Ha, J., and Mclaughlin, S. W., "On rate-adaptability of nonbinary LDPC codes," in *Proc. 5th Intern. Conf. on Turbo Codes and Related Topics*, (Lausanne, Switzerland), Sep 2008.

[39] Klinc, D., Ha, J., and Mclaughlin, S. W., "Optimized puncturing and shortening distributions for nonbinary LDPC codes over the binary erasure channel," in *Proc. Allerton Conference on Communication, Control, Computers*, (Monticello, IL), Oct 2008.

[40] Klinc, D., Ha, J., and Mclaughlin, S. W., "Puncturing and shortening non-binary LDPC codes," *journal paper in preparation*, 2011.

[41] Klinc, D., Ha, J., Mclaughlin, S. W., Barros, J., and Kwak, B.-J., "LDPC codes for physical layer security," in *submitted for publication*, 2009.

[42] Klinc, D., Ha, J., Mclaughlin, S. W., Barros, J., and Kwak, B.-J., "LDPC codes for physical layer security," in *Proc. Global Telecommunications Conference (GLOBECOM)*, (Honolulu, HI), Dec 2009.

[43] Klinc, D., Ha, J., Mclaughlin, S. W., Barros, J., and Kwak, B.-J., "LDPC codes for the gaussian wiretap channel," in *Proc. Information Theory Workshop (ITW)*, (Taorimina, Italy), Oct 2009.

[44] Klinc, D., Ha, J., Mclaughlin, S. W., Barros, J., and Kwak, B.-J., "LDPC codes for the gaussian wiretap channel," *accepted for publication in IEEE Transactions on Information Forensics and Security*, 2011.

[45] Klinc, D., Hazay, C., Jagmohan, A., Krawczyk, H., and Rabin, T., "On compression of data encrypted with block ciphers," *submitted to IEEE Trans. Info. Theory.*

[46] Klinc, D., Hazay, C., Jagmohan, A., Krawczyk, H., and Rabin, T., "On compression of data encrypted with block ciphers," in *Proc. Data Compression Conference*, (Snowbird, UT), Mar 2009.

[47] Kwon, J., Klinc, D., Ha, J., and Mclaughlin, S. W., "Fast decoding of rate-compatible punctured LDPC codes," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, (Nice, France), Jun 2007.

[48] Liveris, A., Xiong, Z., and Georghiades, C., "Compression of binary sources with side information at the decoder using LDPC codes," *IEEE Communications Letters*, vol. 6, pp. 440–442, Oct 2002.

[49] Luby, M., Mitzenmacher, M., Shokrollahi, M., and Spielman, D., "Efficient erasure correcting codes," *IEEE Transactions on Information Theory*, vol. 47, pp. 569–584, Mar 2001.

[50] MacKay, D. J. C., "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, pp. 399–431, Mar 1999.

[51] Mahdavifar, H. and Vardy, A., "Achieving the secrecy capacity of wiretap channels using polar codes," in *Proc. International Symposium on Information Theory (ISIT)*, (Austin, TX), 2010.

[52] Mao, W., *Modern Cryptography: Theory and Practice.* Prentice Hall, 2003.

[53] Massey, J. L., "Guessing and entropy," in *IEEE Inter. Symp. on Info. Theory*, (Seattle, WA), June 1994.

[54] of Commerce/National Institute of Standards, U. D. and Technology, "Advanced encryption standard (AES)," in *FIPS PUB 197*, Nov. 2001.

[55] of Standards, N. B., *Data Encryption Standard (DES).* U.S. Department of Commerce, Washington D.C, 1977.

[56] Ozarow, L. and Wyner, A. D., "Wire-tap channel II," *AT&T Bell Laboratories technical journal*, vol. 63, pp. 2135–2157, Dec 1984.

[57] Park, H. Y., Kang, J. W., and Kim, K. S., "Efficient puncturing method for rate-compatible low-density parity-check codes," *IEEE Transactions on Wireless Communications*, vol. 6, pp. 3914–3919, Nov 2007.

[58] Pishro-Nik, H. and Fekri, F., "Results on punctured low-density parity-check codes and improved iterative decoding techniques," *IEEE Transactions on Information Theory*, vol. 53, pp. 599–614, Feb 2007.

[59] Rathi, V. and Urbanke, R., "Density evolution, thresholds and the stability condition for non-binary ldpc codes," *IEE Proceedings Communications*, vol. 152, pp. 1069–1074, Dec 2005.

[60] Richardson, T. and Urbanke, R., "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Transactions on Information Theory*, vol. 47, pp. 599–618, Feb 2001.

[61] Richardson, T. and Urbanke, R., "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, pp. 619–637, Feb 2001.

[62] Richardson, T. and Urbanke, R., "Efficient encoding of low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, pp. 638–656, Feb 2001.

[63] Richardson, T. and Urbanke, R., *Modern Coding Theory.* Cambridge University Press, 2008.

[64] Shannon, C., "Communication theory of secrecy systems," *Bell Systems Technical Journal*, vol. 29, pp. 656–715, Jan 1949.

[65] Shi, C. and Ramamoorthy, A., "Design and analysis of e2rc codes," *IEEE Journal on Selected Areas in Communications*, vol. 27, pp. 889–898, Aug 2009.

[66] Slepian, D. and Wolf, J., "Noiseless coding of correlated information sources," *IEEE Trans. Info. Theory*, vol. 19, pp. 471–480, July 1973.

[67] Storn, R. and Price, K., "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, pp. 341–359, Dec 1997.

[68] Tanner, M., "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, pp. 533–547, Sep 1981.

[69] Thangaraj, A., Dihidar, S., Calderbank, A. R., McLaughlin, S. W., and Merolla, J. M., "Applications of ldpc codes to the wiretap channels," *IEEE Transactions on Information Theory*, vol. 53, pp. 2933–2945, Aug 2007.

[70] Tian, T. and Jones, C., "Construction of rate-compatible ldpc codes utilizing information shortening and parity puncturing," *EURASIP Journal on Wireless Communications and Networking*, no. 5, 2005.

[71] Vellambi, B. N. and Fekri, F., "Finite-length rate-compatible ldpc codes: A novel puncturing scheme," *IEEE Transactions on Communications*, vol. 57, pp. 297–301, Feb 2009.

[72] Wymeersch, H., Steendam, H., and Moeneclaey, M., "Log-domain decoding of ldpc codes over gf (q)," in *Proc. IEEE International Conference on Communications*, 2004.

[73] Wyner, A. D., "The wire-tap channel," *The Bell System Technical Journal*, vol. 54, pp. 1355–1387, Oct 1975.

[74] Yeo, E., Pakzad, P., Nikolic, B., and Anantharam, V., "High throughput low-density parity-check decoder architectures," in *Proc. IEEE International Conference on Communications*, pp. 3019–3024, 2001.

[75] Yue, G., Wang, X., and Madihian, M., "Design of rate-compatible irregular repeat accumulate codes," *IEEE Transactions on Communications*, vol. 55, pp. 1153–1163, Jun 2007.