On Compression of Encrypted Data

by

Mo Deng

B.S. in Electrical Engineering (with the highest honor), University of Illinois Urbana Champaign(2013)
B.S. in Mathematics (with the highest honor), University of Illinois Urbana Champaign(2013)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2016

Signature redacted

Author . . . .
Department of Electrical Engineering and Computer Science
May 20, 2016

Signature redacted

Certified by . . . .
Gregory W. Wornell
Sumitomo Professor of Engineering
Thesis Supervisor

Signature redacted

Accepted by . . . .
Leslie A. Kolodziejski
Chairman, Department Committee on Graduate Theses

On Compression of Encrypted Data

by

Mo Deng

Submitted to the Department of Electrical Engineering and Computer Science
on May 5, 2016, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering

Abstract

In this thesis, I took advantage of a model-free compression architecture, where the encoder only makes decision about coding and leaves to the decoder to apply the knowledge of the source for decoding, to attack the problem of compressing encrypted data. Results for compressing different sources encrypted by different class of ciphers are shown and analyzed. Moreover, we generalize the problem from encryption schemes to operations, or data-processing techniques. We try to discover key properties an operation should have, in order to enable good post-operation compression performances.

Thesis Supervisor: Gregory W. Wornell
Title: Sumitomo Professor of Engineering

Acknowledgments

First, I would like to express my sincere gratefulness to my research supervisor, Prof. Gregory Wornell, for his guidance and insights. Moreover, for most of the past two years and unfortunately even until today, I couldn't find much passion in doing research. At some point, I started to doubt the choices of going to graduate school that I made for myself a couple years ago. My life hasn't been ignited for long and it was hard to for me to be "mentally engaged"(as Greg once pointed out to me) in my work, though I believed I tried my best. Often times, I feel guilty of not able to live up to the expectations of Greg, and many others. Therefore, I am truly grateful that Greg keeps patient and still offers me a position in the group. I would also like to thank Tricia for her efforts to put the group together and all SIA students and visitors that I have met since my arrival, for their insightful discussions, constant encouragements as well as sharp but constructive criticisms.

Moreover, I would like to thank my family, and a long list of friends, for their unconditional supports, especially when I went through some of the unprecedented difficulties in my personal life. With all of you, Boston has always been a warm place.

Contents

1 Introduction **1**

1.1 Motivation 3

2 Probabilistic Graphical Models and Model-Free Compressions **7**

2.1 Markov Random Fields 8

2.1.1 Conditional Dependencies from UGMs 9

2.1.2 Factor Graphs, Parametrization of MRFs 10

2.1.3 Examples of MRFs 11

2.2 LDPC-based Model-free Compression 14

2.2.1 LDPC codes 14

2.2.2 A (Binary) Model-Free Compression Architecture 15

2.3 Generalization of the Architecture: 20

3 Useful Cryptosystems **23**

3.1 One-Time-Pad(OTP) 23

3.2 RSA Cryptosystem 25

3.3 McEliece Cryptosystem 27

4 Proposed Architecture and Simulation Results **31**

4.1 Source Specifics: 31

4.2 Compression of Encrypted Data: 35

4.2.1 An algorithm of compressing encrypted data and its performance in theory: 36

4.2.2 Simulation Results on Synthetic Data 37
4.2.3 Compressing Encrypted text files 48
4.3 Discussions 52
4.3.1 Discussions on the results 52
4.3.2 On finding the (sub-)optimal encoding matrix H 60
4.3.3 On optimal choice of doping 64
4.3.4 On Compression based on non-binary LDPC codes 65

List of Figures

1-1 Conventional system(Figure from [**1**]) 2

1-2 Compressing Encrypted Data(Figure from [**1**]) 2

2-1 Undirected Graphical Model(Figure from [**10**]) 10

2-2 Factor Graph. (Figure from [**39**]) 11

2-3 Markov Chain of Order 1(Figure from [**37**]) 12

2-4 2D Ising Model with Periodic Boundary Conditions(Figure from [**37**]) 13

4-1 Entropy of Ising models of different sizes 35

4-2 Compression of OTP-Encrypted Binary Bernoulli Sources 39

4-3 Compression of OTP encrypted Binary Markov source, doping rate=0 40

4-4 Compression of OTP encrypted Binary Markov source, doping rate=0.1 41

4-5 Compressing OTP-Encrypted 10×10 Ising Model 42

4-6 Compressing OTP-Encrypted 100×100 Ising Model 43

4-7 Compressing encrypted 4-bit Markov sources,doping=0.05 46

4-8 Compressing encrypted 4-bit Markov sources,doping=0.10 47

4-9 Compressing encrypted 8-bit Markov sources,doping=0.05 48

4-10 Compressing encrypted 8-bit Markov sources,doping=0.10 49

4-11 Girth of PEG Tanner Graph 63

List of Tables

3.1 Large Alphabet One Time Pad Encryption 24

4.1 Compression of Classical File 1 51
4.2 Compression of Classical File 2 51
4.3 Compression of Encrypted Economist files 52
4.4 Compression of Encrypted Synthetic Markov Data for English 52
4.5 (4,7) McEliece Encryption 55
4.6 (4,7) RSA Encryption 57

Chapter 1

Introduction

Traditionally in communication systems, data from a source are first compressed and then encrypted before being transmitted over a channel to the receiver(Figure 1-1). However, there exist some scenarios where there is a need to reverse the order of the two, i.e. the data should first be encrypted and then compressed[**1**][**2**], before being transmitted to the channel. For example, the owner of the data may not trust the party which conducts the compression, so that he wishes the data to be encrypted before it is compressed, without having to compromise the compressibility of the data. However, since the encrypted data seems random and thus any conventional compression techniques, which in general relies on taking advantage of the knowledge of some underlying statistical structures of the source when designing the encoder, did not yield desirable results in practice. This thesis is aimed at attacking this problem. To be more precise, we investigate the possibility of achieving as good compressibility of the data when it has been encrypted by some practical schemes as when no encryption is performed at all. Besides, since any encryption scheme can be seen as a kind of data processing technique,, we also aim at discovering the key properties of data processing techniques that could enable good post-processing compression performances.

The thesis is organized as follows: Chapter 2 will introduce the necessary backgrounds for understanding various probablistic source models and the model-free compression that will be used later. Chapter 3 reviews different encryption schemes involved. In

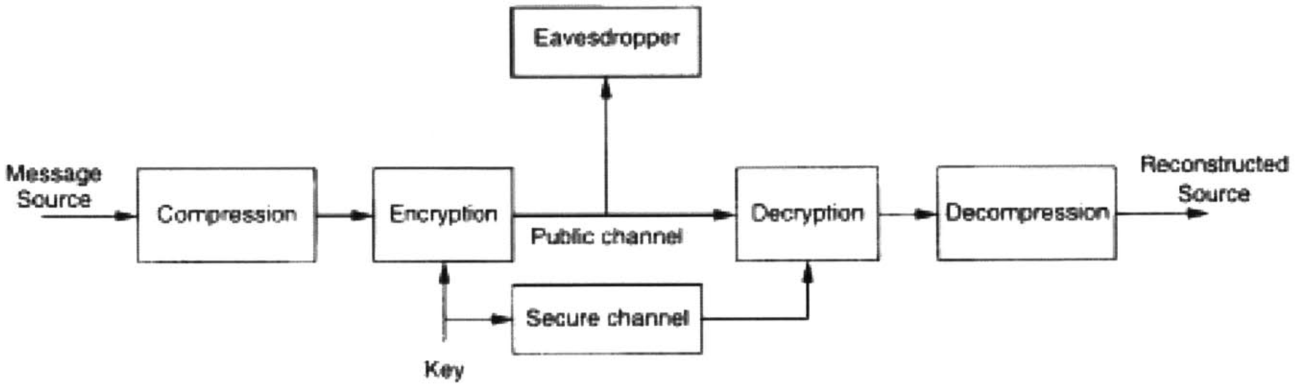

Figure 1-1: Conventional system(Figure from [**1**])

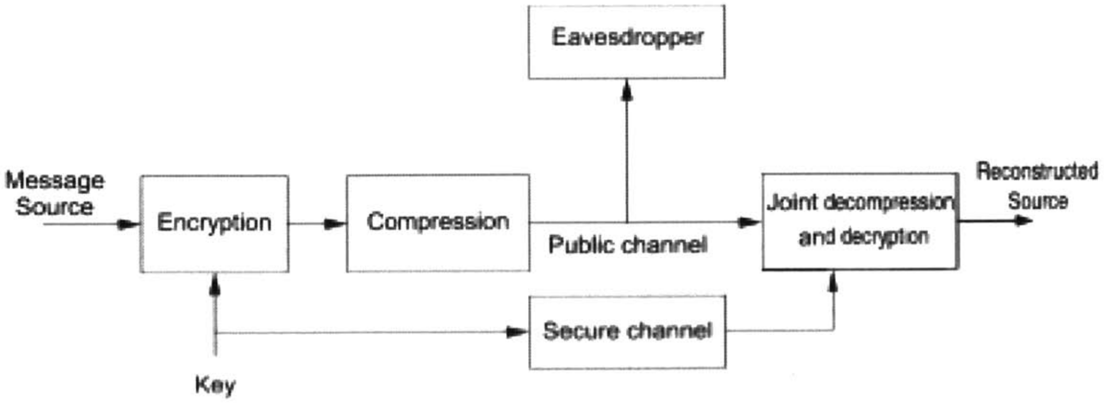

Figure 1-2: Compressing Encrypted Data(Figure from [**1**])

Chapter 4, the architecture of compressing encrypted data will be proposed. We will also argue the legitimacy of such architecture in theory and show simulation results of compression performances of some classical sources encrypted by various encryption schemes and discuss the reasons why performances are significantly different for different sources and encryption schemes. Moreover, we will extend the discussion in Chapter 5 and try to characterize the general properties that data-processing techniques should have to achieve desirable compression performance. Finally, Chapter 6 points out some open problems and concludes the thesis.

1.1 Motivation

Data compression, also referred to as *coding*, is used everywhere[**3**]. Most images available on the web are compressed, typically into JPEG or GIF formats; many file systems automatically compress files when stored and so on. It is useful because it helps reduce resource usage, such as data storage space or transmission capacity.

In the context of compression, the generic term *message* are often used for the objects to be compressed. The compression system consists of two components, an *encoder* that takes a message and generates a "compressed" representation, which is usually shorter than the original message, and a *decoder* that reconstructs the original message or some approximation of it from the compressed representation. The former is called *lossless* compression and the latter is called *lossy* compression. The encoder and the decoder are typically intricately tied together since often times, they both have to understand the shared compressed representation.

Lossless compressions and lossy compressions are of interest for different applications. Typically, lossless compression schemes are used for texts and lossy ones for images and videos, where a little bit loss in resolution is often undetectable and thus acceptable. In this thesis, unless otherwise specified, the word *compression* refers to *lossless* *compression.*

Typically, lossless compression, which essentially tries to encode information using fewer bits than the original representation, without any loss of information so that a well-designed decoder may successfully reconstruct the original representation, is not achievable unless some redundancy underlying the probabilistic structure of the source can be well exploited by the compression architecture. Such probabilistic structure is often referred to as the *data model* or *source model*, which we will use interchangeably throughout the thesis. Such source model determines some "bias" in the input messages, i.e., some inputs are more likely than others, which can be taken advantage of to enable more efficient representation of the source.

Claude Shannon borrowed the definition of *entropy* [**4**][**5**]from statistical physics, where entropy represents the randomness of a system, to the context of information theory, where (Shannon) entropy is used to measure the uncertainty of the source. Shannon also introduced two *source coding theories* for data compression, one for lossless compression and the other for lossy compression(a.k.a. rate distortion theory[**33**]). Specifically for loseless compression, the source coding theory states that it is impossible to compress the data such that the code rate, defined as the average number of bits per symbol, is less than the Shannon entropy of the source, without it being virtually certain that some information in the source will be lost. However, it is possible to get the code rate arbitrarily close to the Shannon entropy, with negligible probability of loss of information. The source coding theory, however, does not indicate what such an entropy-approaching compression architecture is, for a given source.

When redundant data are transmitted over a communication channel, which is often deemed bandwidth-constrained and insecure, needs for both compression and encryption arise. Since any good encryption scheme tends to make the ciphertexts look completely random so as to increase the security, the compressibility seems to have been compromised. Thus, it has long been natural to first compress the data according to the source model and encrypt the compressed stream and at the receiver, decryption is conducted, followed by the decompression process.

However, in some applications, reversing the order of the two may be necessary: for example, the owner of the data may not trust the party who does the compression. Thus, it is desirable if data is encrypted before it is sent to the compressor. It motivates us to reflect on the possibility of compressing the encrypted data. This research is initially motivated by the goal of compressing a classified English file, which has been encrypted.

We note that with a key of encryption given, the mapping between plaintexts and ciphertexts must be one-to-one to make the encryption scheme valid. Therefore, the encryption process does not change the conditional entropy(on the key used) of the source, therefore, according to the source coding theory, the compressibility of the data should remain the same, although it is likely hard to explicitly express the model underlying the encrypted data, even if the key is known, which makes it hard to come up with good compression schemes to compress those encrypted data.

Recently, a model-free coding system was established[**6**] [**7**], where the encoder only makes choices about coding and produces a compressed bit-stream agnostic to the source model. It instead leaves to the decoder to apply the relevant knowledge about the source. Under this structure, no information about the source needs to be released to the encoder, which can help the data owner retain total secrecy. We note that our problem of interest, compressing of encrypted data, may be enabled by this structure.

The aforementioned compression architecture was based on graphical models and iterative decoding. At the encoder, the source symbols(in the form of their binary representations) are compressed by a parity check matrix of a LDPC(Low Density Parity-Check) code. The decoder, composed of a source subgraph as well as a code subgraph, runs (loopy) belief propagation on the combined source-code graph, where source nodes are shared between the source and code subgraphs. Messages are passed along all edges in the graph, according to the standard sum-product algorithm.The algorithm runs until (hopefully) convergence or declaration of failure. Final decoding is done based on standard belief equation, where the belief of each source symbol is the combination of partial beliefs from the two subgraphs. More details about the model-free compression architecture will be covered in Chapter 2.

Note that, belief propagation(or sum-product) algorithms, which become popular means of solving inference problems, are exact only for tree-structured graphical models. In cases where there are short cycles in the graphical models, it may not converge but if it does, it usually provides good approximate solutions to the problem. Moreover, if the approximation is indeed good, the approximation nature is often a more than acceptable price for performing efficient inference. In [**8**][**9**], sufficient conditions are given for a loopy BP to converge to good approximate solutions, however, these conditions are not readlily applicable. Therefore, we will run simuation results in Chapters 4 and 5 to understand the empirical performances.

Chapter 2

Probabilistic Graphical Models and Model-Free Compressions

As mentioned before, some probabilistic structure in the data source needs to be well exploited by the compressor to achieve good compression performances. Intuitively, to understand the inherent structures of the source, it is worth seeing how the joint distribution, or more generally, a global function, can be factored into a product of several local functions, with each of which having a subset of the variables as its arguments.

If we constrain the global function to be the joint probability density function of the information source, then this factorization essentially encodes the conditional dependent structure among the source variables and can be conveniently expressed via a graph, called the *probabilistic graphical model*(PGM)[**34**]. Later in the thesis, the source is compactly described via such a graph, called the *source subgraph.* In section 2.1, we will give a more detailed description on PGM, with the emphasis on a branch of PGM, *Markov Random Fields*, which all sources of interest in the thesis can be categorized into.

A useful class of variations, *factor graphs*, is a class of undirected bipartite graphs connecting variables and factors, where each factor represents a function over the variables it is connected to. If the global function is again the probability density function of the source, then the factor graph describing the source is essentially equivalent to the PGM description of the same source in the sense that each factor node in the factor graph corresponds to a maximal clique in the PGM, with the variables connected to each factor node being the ones that consists the maximal clique. In this thesis, we use the PGM to describe the source subgraph but nevertheless introduce the factor graph as a seperate topic since the LDPC code(low denstiy parity-check code), on which model-free compression structure is based, is defined based upon factor graphs. More descriptions on factor graphs, LDPC codes and LDPC code-based, model-free compression architecture are given in sections 2.2 ad 2.3, respectively.

2.1 Markov Random Fields

There are two most common branches of PGMs, *Bayesian networks* and *Markov Networks*, also known as *Markov random fields*(MRFs)[**10**][**11**]. The underlying semantics of Bayesian networks are based on directed graphs so that Bayesian networks are also called directed graphical models; on the other hand, the underlying semantics of MRFs are based on undirected graphs and thus MRF are also called *undirected graphical models*(UGMs). In this thesis, all source models of interest are MRFs so that we will only introduce MRFs hereafter.

The graphical structure of an undirected graph represents some of the qualitative properties of the joint distribution of the source variables – the nodes in the graph of an MRF represent the variables and the edges correspond to some notion of direct probabilistic interactions between neighbouring variables. In next subsection, we will introduce the conditional dependence properties encoded in the PGMs.

2.1.1 Conditional Dependencies from UGMs

Typically, we use a graph $G = (V, E)$, with a set of random variables $S = (S_v)_{v \in V}$ indexed by elements in V, to describe an MRF with respect to G. The construction of edges must obey the statistical structure of the source and the following properties can be read from the UGM:

Pairwise Markov Property – an edge between two variable nodes S_i and S_j is **absent** in the graph if and only if S_i and S_j are **conditionally independent**, given all the other variables in the graph. Note that, it is only the **absence** of edges that makes a graphical representation useful for describing the distribution. In Figure 2-1, for example, 1 and 5 are conditionally independent, given all other variables. We denote this relation as $1 \perp 5 \mid \{2, 3, 4, 6, 7\}$.

Local Markov Property – any arbitrary source node s_i, $i \in V$, given its neighbourhood on G , $\mathcal{N}(i)$, is independent of all other variables.
Namely, $p(s_i \mid s_{V \sim i}) = p(s_i \mid s_{\mathcal{N}(i)})$ for all $i \in V$. For example, in Figure 2-1, $1 \perp \{4, 5, 6, 7\} \mid \{2, 3\}$.

Global Markov Property – for any disjoint subsets of nodes A, B and C in the UGM graph G such that C separates A and B (i.e. every path between a node in A and a node in B passes through a node in C), the random variables S_A and S_B are conditionally independent, given S_C. For example, in Figure 2-1, we can see that, $\{1, 2\} \perp \{6, 7\} \mid \{3, 4, 5\}$.

Note that in general, the Local Markov Property is weaker than the Global Markov Property, while stronger than the Pairwise Markov Property. However, for any strictly positive distributions, those three properties are equivalent.

We have mentioned that a given source can also be described via a factor graph.

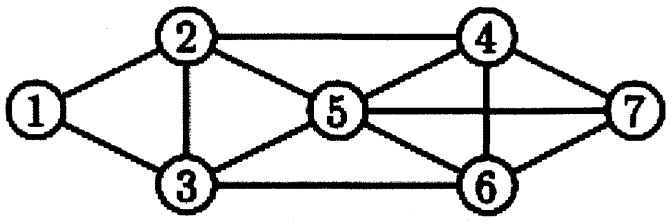

Figure 2-1: Undirected Graphical Model(Figure from [10])

Although we will not use the factor graph to explicitly describe the source subgraph, for the purposes of introducing the LDPC codes later, we will briefly introduce factor graphs in the next subsection.

2.1.2 Factor Graphs, Parametrization of MRFs

Suppose that a global function $g(x_1, \ldots x_n)$ factors into a product of several local functions f_j's, each having its argument as some subset of $\{x_1, \ldots x_n\}$, i.e.

$$g(x_1, \ldots x_n) = \prod_{j \in \mathcal{J}} f_j(X_j) \tag{2.1}$$

where $\mathcal{J}$ is a discrete index set, X_j is a subset of $\{x_1, \ldots x_n\}$ and $f_j(X_j)$ is a function having the elements of X_j as arguments. With that, we can define the corresponding factor graph as follows:

Definition. A *factor graph* is a bipartite graph that expresses the structure of the factorization in (2.1). A factor graph has a *variable node* for each variable x_i, a *factor node* for each local function f_j. An edge connects variable node x_i and factor node f_j if and only if x_i is an argument of f_j. Figure 2-2 gives an example of a simple factor graph, where

$$g(x_1, x_2, x_3, x_4, x_5) = f_A(x_1) f_B(x_2) f_C(x_1, x_2, x_3) f_D(x_3, x_4) f_E(x_3, x_5), \tag{2.2}$$

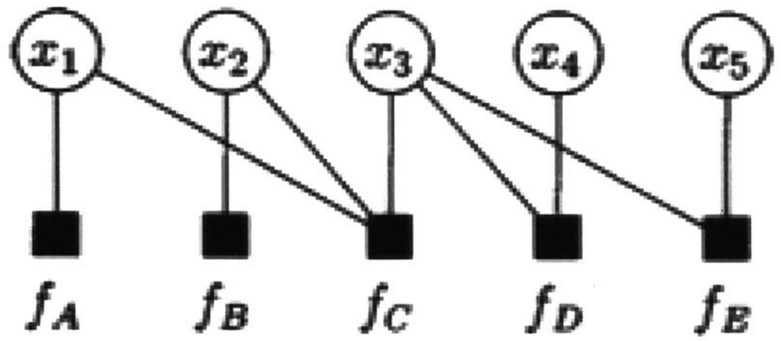

Figure 2-2: Factor Graph. (Figure from [**39**])

As mentioned before, though in different forms, there are close connections between factor graphs and graphical models for MRFs – the joint probability mass function of an MRF with graphical model $\mathcal{G} = (S = \{s_1, \ldots, s_n\}, E)$, $p(s)$, can be factored over maximal cliques of $\mathcal{G}$:

$$p(s) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \phi_C(s_C), \tag{2.3}$$

where Z is a normalizing factor to make sure that $p(s)$ is a valid p.m.f.

Apparently, each maximal clique C in the UGM of an MRF corresponds to a factor in the factor graph. Moreover, the weight associated with the maximal clique C, $\phi_c(s_C)$, can be seen as the local function corresponding to the maximal clique C, with its arguments as the variables in the maximal clique C.

2.1.3 Examples of MRFs

Now we give several examples of MRFs, all of which are used later in the thesis.

(a) Markov Chains:

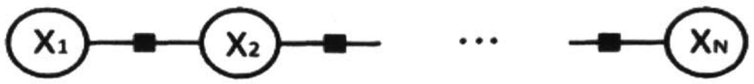

Figure 2-3: Markov Chain of Order 1(Figure from [**37**])

Markov Chain is the simplest example of MRF. A sequence of random variables $X = (x_1, \ldots, x_n)$ is an k^{th} order Markov chain if for $\forall i > k$,

$$P(x_i \mid x_{i-1}, \ldots, x_1) = P(x_i \mid x_{i-1}, \ldots x_{i-k}), \tag{2.4}$$

We will later see English texts can be modeled as the first and second order Markov chains, in Chapter 4. Figure 2-3 shows the schematic(graphical model) of a first order Markov chain.

Moreover, for a Markov chain of order 1, the joint pmf of the source can be factored as:

$$p(x) = p(x_1, \ldots, x_n) = p(x_n \mid x_{n-1})p(x_{n-1} \mid s_{x-2}) \ldots p(x_2 \mid x_1)p(x_1), \tag{2.5}$$

which clearly has the of the form of MRF. Each pair of two neighbouring nodes forms a maximal clique of the graph.

(b) 2D Ising Model:

Ising Model[**12**] was originally a mathematical model of ferromagnetism in statistical mechanics, named after the physicist Ernst Ising. It consists of discrete variables that represent magnetic dipole moments of atomic spins that can be in one of the two states.(+1, -1). Spins are arranged in a lattice-like graph, allowing each spin to interact with its neighbours. The two-dimensional Ising model is the simplest one to show a phase transition. Later, it is used widely as an image model in computer

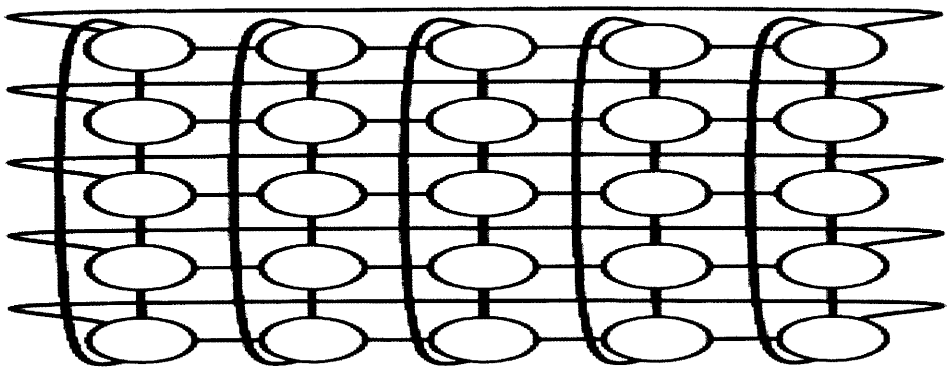
Figure 2-4: 2D Ising Model with Periodic Boundary Conditions(Figure from [37])

vision research, where each source node represents a pixel and takes the value of 0 or 1, as will be in our case.

Definition: The homogeneous Ising source $s^{h\times w}$ is defined over the $\{0,1\}^{h\times w}$ lattice graph $\mathcal{G} = (\mathcal{S}, \mathcal{E})$, by

$$p(s^{h\times w}) = \frac{1}{Z}\prod_{i\in\mathcal{S}}\phi_i(s_i)\prod_{(i,j)\in\mathcal{E}}\varphi_{ij}(s_i,s_j), \tag{2.6}$$

where for each $i \in \mathcal{S}, \phi_i(1) = 1 - \phi_i(0) = p_{bias}$, as the node potentials. And for each $(i,j) \in \mathcal{E}$,

$$\varphi_{ij}(s_i,s_j) = \begin{cases} p_{stay} & \text{if } s_i = s_j \\ 1 - p_{stay} & \text{if } s_i \neq s_j \end{cases} \tag{2.7}$$

Figure 2-4[37]shows the schematic of the 2D Ising Model.

A few remarks:

(a) 2D Ising model can be seen as the 2D extension of a binary Markov chain.

(b) In this thesis, we will only consider Ising models that are homogeneous and symmetric about 0 and 1. Homogeneous here means the node and edge potentials are defined hemogeneously over the whole grid and symmetric about 0 and 1 are guaranteed by both $p_{bias} = 0.5$ as well as $\varphi_{ij}(0,1) = \varphi_{ij}(1,0) = 1 - p_{stay} = 1 - \varphi_{ij}(0,0) = 1 - \varphi_{ij}(1,1)$, for $\forall (i,j) \in \mathcal{E}$.

(c) 2D Ising model is assumed to have **periodic boundary conditions**, which is self-evident, with the help of Figure 2-4.
(d) 2D Ising sources cannot be sampled exactly. For simulation purposes, in this thesis, we will use the Gibbs sampled Ising sources of burn-in period 1000, which is a widely used approximate. More detailed descriptions of Gibbs sampling can be found in [**20**].

2.2 LDPC-based Model-free Compression

2.2.1 LDPC codes

This section introduces LDPC codes and some related definitions to prepare for the LDPC-based model-free compressions, which will come up afterwards.

LDPC codes[**13**][**14**][**25**], or low-density parity-check codes, are a class of linear block codes which provide near-capacity performance on a large set of data-transmission and data-storage channels. Here, we will only consider binary LDPC codes, though brief discussions on non-binary LDPC based compression schemes will be given in Chapter 4. For binary LDPC codes, the codebook $\mathcal{C}$ is a subset of $\{0,1\}^n$, with n being the block length of the code, that can be written compactly as

$$\mathcal{C} = \{y \in \{0,1\}^n : Hy = 0 \text{ (mod 2)}\}, \tag{2.8}$$

where H is a binary $k \times n$ parity-check matrix describing this code. To qualify as a low-density parity-check matrix, H is sparse, namely, the number of 0's in H is much larger than that of 1's. Number of 1's is allowed to grow at most linearly with the block length n.

Each low-density parity-check matrix is also uniquely associated with a factor graph with each column corresponding to a variable node while each row corresponding to a factor node(a parity-check node, or check node). We will be using terms factor nodes and check nodes interchangebly in the thesis. In the factor graph, variable node i is connected to check node a if and only if $\mathrm{H}_{a,i} = 1$. For a particular check node a, the number of 1's in the corresponding row of H represents the number of variables nodes connected with a in the associated factor graph. Similarly, for each source node i, the number of 1's in the corresponding column of H represents the number of check nodes connected with i in the associated factor graph.

If each row has same number of 1's ,w_r, and each columns has the same number of 1's, w_c, then the LDPC code is called a *regular* LDPC code, satisfying $kw_r = nw_c$. Otherwise, it is called an *irregular* LDPC code.

The rate of the code is defined as:

$$r_{code} = \frac{\text{rank(H)}}{n} = \frac{\text{number of independent rows of H}}{n} \leq r_{nom} = k/n, \qquad (2.9)$$

where we have defined, k/n, as the nominal compression rate, or r_{nom} of the code.

2.2.2 A (Binary) Model-Free Compression Architecture

Later in the thesis, we will investigate the possibility of achieving good compression rate after the source data has been encrypted by practical encryption schemes. As will become clear later, the theoretical legitimacy of the proposed algorithm partly comes from the recent development of the model-free architecture for compression[**6**][**7**], in which the compressor can only make choices about coding and produce a compressed bit-stream regardless of the source model– in fact, the compressor does not need to have the knowledge of the source model. It leaves to the decoder to apply the knowledge about the source to conduct the decompression. Therefore, this compression architecture may help resolve the difficulty of not knowing any information about the encrypted stream at the compressor and the decoder, with the knowledge of the source model and the key used, may conduct the decryption and decompression jointly.

Now, we give a in-depth description of the model-free compression architecture. Many of the following descriptions are from[6][7]. Interested readers are referred to those articles for further information:

Basic Notations: Let s be an binary vector of length n, drawn from a Markov random field(MRF) $p(s)$. This is the source to be compressed. H is an $k \times n$ (binary) LDPC parity check matrix.

Encoding: Regardless of what $p(s)$ is, H is used as the encoding matrix and the compression is done by:

$$x = Hs, \tag{2.10}$$

where again, the multiplication is in the sense of modulo 2 sum.

Decoding:

(i) *The code subgraph*: Let $S = \{s_1, \dots, s_n\}$ denote the set of source nodes and $X = \{x_1, \dots, x_k\}$ denote the set of compressed bits, or in the context of factor graphs, the set of all factor(check) nodes. Furthermore, let $\mathcal{F}$ denote the set of all edges in the factor graph. There is an edge between factor node x_a and source node s_i if and only if $H_{a,i} = 1$. Then, the code subgraph can be compactly represented by the factor graph $\mathcal{C} = (S \cup X, \mathcal{F})$. H enforces k hard constraints, each of the form $x_a = \sum_{i \in S} H_{a,i} s_i$, for $\forall a \in \{1, 2, \dots, k\}$.

It is worth pointing out, since the complexity of inference(or decoding) on $\mathcal{C}$ scales with the number of edges between source and factor nodes, and in LDPC, where the number of 1's in the parity check matrix H grows at most linearly with n. Therefore, the inference complexity grows linearly with the source length, which is more than acceptable. Low complexity in decoding is one of the most important advantage for

LDPC based compression.

(ii) *The source subgraph:* Recall that, the MRF can be factored with respect to the maximal cliques in the source subgraph. However, the complexity of inference on $\mathcal{G}$ depends on the number of cliques and their sizes[**36**]. Thus, it is desirable if the maximal cliques have smaller cardinality. For now, we will only consider a subset of MRF that assumes a *pairwise factorization*, such that:

$$p(s) = \frac{1}{Z}\prod_{i\in\mathcal{S}}\phi_i(s_i)\prod_{(i,j)\in\mathcal{E}}\varphi_{ij}(s_i,s_j), \tag{2.11}$$

where there is a node potential $\phi_i(s_i)$ associated with each source node and there is an edge potential $\varphi_{ij}(s_i,s_j)$ associated with each neighbouring nodes pair $(i,j)\in\mathcal{E}$ in source subgraph $\mathcal{G}$. Note that, both Markov Chain of order 1 and Ising model assume this factorization.

(iii) *Decoding algorithm*:
Notation: Let $m^{(t)}_{a\to i}(s_i)$ be the message from the factor node x_a to the source node s_i; $m^{(t)}_{k\to i}(s_i)$ be the message from a source node s_k to its source subgraph neighbour s_i; $m^{(t)}_{i\to a}(s_i)$ be the message from source node s_i to factor node x_a, all at time t. Also, let $\mathcal{N}^{\mathcal{G}}_i$ be the set of neighbouring source nodes of source node s_i. Similarly, let $\mathcal{N}^{\mathcal{C}}_i$ be the set of factor node neighbours of source node s_i and let $\mathcal{N}^{\mathcal{C}}_a$ be the set of source node neighbours of factor node x_a. By the standard sum-product algorithms, we have:

(1) **Source Subgraph Message Update:**

$$m^{(t+1)}_{i\to j}(s_j) = \sum_{s_i}\Big[\prod_{x_a\in\mathcal{N}^{\mathcal{C}}_i} m^{(t)}_{a\to i}(s_i)\Big]\phi_i(s_i)\varphi_{ij}(s_i,s_j)\prod_{s_k\in\mathcal{N}^{\mathcal{G}}_i\backslash s_j} m^{(t)}_{k\to i}(s_i), \tag{2.12}$$

where the term in the bracket is the code subgraph belief at s_i, which works as an external message inserted into the source subgraph via s_i.

(2)**Code Subgraph Message Update:**

Source to Factor Message Updates:

$$m_{i\to a}^{(t+1)}(s_i) = \Big[\prod_{s_j\in\mathcal{N}_i^{\mathcal{G}}} m_{j\to i}^{(t)}(s_i)\phi_i(s_i)\Big] \prod_{x_b\in\mathcal{N}_i^{\mathcal{C}}\backslash x_a} m_{b\to i}^{(t)}(s_i), \tag{2.13}$$

where the term in the bracket is the source subgraph belief at s_i, which works as an external message inserted into the code subgraph via s_i.

Factor to Source Message Updates:

$$m_{a\to i}^{(t+1)}(s_i) = \sum_{\mathcal{N}_a^{\mathcal{C}}\backslash s_i} f_a(\mathcal{N}_a^{\mathcal{C}}) \prod_{s_j\in\mathcal{N}_a^{\mathcal{C}}\backslash s_i} m_{j\to a}^{(t)}(s_j), \tag{2.14}$$

where $f_a(\mathcal{N}_a^{\mathcal{C}})$ is the indicator for the hard constraint imposed by x_a, and is defined as:

$$f_a(\mathcal{N}_a^{\mathcal{C}}) = \begin{cases} 1 & \text{if constraint } x_a = \sum_{i\in S} H_{a,i}s_i \text{ is satisfied} \\ 0 & \text{if otherwise} \end{cases} \tag{2.15}$$

(3)**Belief Update:** in each iteration, we can update the belief equation of each source node, according to:

$$\hat{s}_i^{(t)} = \arg\max_{s_i} \Big[\prod_{x_a\in\mathcal{N}_i^{\mathcal{C}}} m_{a\to i}^{(t)}(s_i)\Big]\Big[\prod_{s_j\in\mathcal{N}_i^{\mathcal{G}}} m_{j\to i}^{(t)}(s_i)\phi_i(s_i)\Big], \tag{2.16}$$

where the belief of a source node should be interpreted as the combination of partial beliefs from the code subgraph and source subgraph, respectively.

Remarks on Doping:

Generally, the decoding process will need to rely on a fraction(hopefully small) of source nodes being directly described to the decoder to converge to the correct solution or a good approximate. This process is called *doping.* In practical, the doping process is realized by augmenting H with additional unit weight rows so that those corresponding source symbols are directly known to the decoder.

Remarks about the implementation of the algorithm:

Several remarks about the simulations to make:

1. The initialization of the messages: messages should be initialized as follows:

at $t = 1$, if x_a is a degree-one factor node(corresponding to a doping source symbol s_i), then source to factor message:

$$m_{i \to a}^{(1)}(\alpha) = \begin{cases} 1 & \text{if } \alpha = s_i \\ 0 & \text{otherwise} \end{cases} \tag{2.17}$$

and the factor to source message:

$$m_{a \to i}^{(1)}(\alpha) = \begin{cases} 1 & \text{if } \alpha = s_i \\ 0 & \text{otherwise} \end{cases} \tag{2.18}$$

if x_a is not a degree-one factor node, then source to factor, factor to source messages are updated as uniform messages over the *source alphabet*, for example, if source is binary, $\forall \alpha \in \{0, 1\}$, $m_{i \to a}^{(1)}(\alpha) = 0.5, m_{a \to i}^{(1)}(\alpha) = 0.5$. For source to source messages, since no source nodes could have any unbiased initial beliefs about other source nodes, source to source messages are initialized as : $m_{i \to j}^{(1)}(\alpha) = 0.5$, $\forall (i, j) \in \mathcal{E}$ and $\forall \alpha \in \{0, 1\}$. From the message updating rules and the message initializations, it is clear that the doping symbols(and the messages involving the doping symbols) can help create dynamics for the convergence of the decoding algorithm.

2. Since the decoder runs the loopy belief propagation, it is essential that short loops are avoided in the combined graph, which includes the code subgraph. Therefore, the encoding matrix H should be constructed so that short-loops(loops of length 4) is avoided. Generally, there are two most common ways to construct H, namely **Gallager construction** and **Progressive Edge Growth(PEG) construction**[**27**]. Generally, Gallager construction provides no guarantee of avoiding short loops, while PEG was developed in order to avoid short loops, or, increase the girth in the underlying graph. Therefore, in this thesis we will use PEG construction. Details of PEG will be included in Chapter 4. Some large PEG constructed matrix are available[**28**].

2.3 Generalization of the Architecture:

If the source data s^n is over a larger alphabet $GF(q)$ for some $q > 2$, we could adapt H into $GF(q)$ and conduct message passing on integers or translate the source into a binary stream and use the binary LDPC based compression. The former approach will be discussed later in the thesis, and we will consider the latter for now. In the latter approach, since there is a symbol translation layer in the source subgraph, intuitively, messages should also be translated between layers. Most of the following descriptions in this section can be found in [6] and [7].

Suppose $s^n = \{s_1, \dots, s_n\}$ is an n-symbol data sequence over $GF(q)$ that is *serialized* by symbol-level representation maps. Without loss of generality, assume all s_i belong to the same alphabet $\mathbf{S}$ of cardinality M. The representation map is a bijection $t_{M\to 2} : \mathbf{S} \to GF(2)^B$ where $B = \lceil \log_2 M \rceil$, representing the length of the translation of each character in the alphabet.

When messages are passed to or from source nodes, there are related messages on their serialized (binary) representations. Define a pair of *message translation functions*, $T_{M\to 2} : (\mathbf{S} \to \mathbf{R}^+) \to (\mathbf{GF}(2) \to \mathbf{R}^+)^B$ and $T_{2\to M} : (\mathbf{GF}(2) \to \mathbf{R}^+)^B \to (\mathbf{S} \to \mathbf{R}^+)$ that converts between a message $m^{(M)}$ over $\mathbf{S}$ and a B-tuple of messages $m^{(2)} = m_1^{(2)}, \dots, m_B^{(2)}$ over $\mathbf{GF}(2)$, such that for $\omega \in \{1, \dots, B\}$ and $\beta \in \{0, 1\}$, and for $\alpha \in \mathbf{S}$:

$$T_{M\to 2}(m^{(M)})_\omega(\beta) = \sum_{\alpha \in S} m^{(M)}(\alpha)\mathcal{I}\{t_{M\to 2}(\alpha)_\omega = \beta\}, \tag{2.19}$$

where $\mathcal{I}\{t_{M\to 2}(\alpha)_\omega = \beta\}$ is the indicator on whether the ω^{th} bit (from the left) of the binary representation of α is equal to β, and

$$T_{2\to M}(m^{(2)})(\alpha) = \prod_{\omega=1}^{B} m_\omega^{(2)}(t_{M\to 2}(\alpha)_\omega), \tag{2.20}$$

If we see each of the B bits of a symbol's binary representation as a random variable, we can see that the message translations are lossless, or equivalently (2.19) and (2.20) are true, if and only if those B bits are independent. **Thus, the proposed**

algorithm does not incur any architectural loss if and only if the B bits in a symbol's binary representation are independent. As can be seen later in the thesis, this independence condition does not always hold and therefore the algorithm may incur some architectural loss, but these conversions can greatly reduce the complexity of the algorithm.

Encoding:

Similar to the binary case, the large alphabet source data is first serialized by $z^{nB} = t_{M\to 2}(s^n)$, then the model-free encoding takes place in the represented alphabet of $\mathbf{GF}(2)$. Choose k to target $r_{code} = k/nB$. Choose a random H of size $k \times nB$ that has k independent rows. and produce the compressed result, $x^k = Hz^{nB}$.

Decoding:

Let $\mathcal{S} = \{s_1, \dots, s_n\}$, the code subgraph is now $\mathcal{C} = (\mathcal{Z}, \mathcal{X}, \mathcal{F})$, where $\mathcal{Z} = t_{M\to 2}(\mathcal{S})$. Denote $\mathcal{N}^{\mathcal{C}}_{i,\omega}$ as the factor node neighbours of $z_{i,\omega} = t_{M\to 2}(s_i)_\omega$. Noting that we always use the messages at iteration t to update messages at iteration $(t+1)$ and we will omit the iteration indices of the messages in the expressions below for simplicity.

Source Message Update:

$$m^{(M)}_{i\to j}(s_j) = \sum_{s_i} \left[m^{(M)}_{\mathcal{C}\to i}(s_i)\right] \phi_i(s_i)\varphi_{ij}(s_i, s_j) \prod_{s_k \in \mathcal{N}^{\mathcal{G}}_i \backslash s_j} m^{(M)}_{k\to i}(s_i), \tag{2.21}$$

where,

$$m^{(M)}_{\mathcal{C}\to i}(s_i) = T_{2\to M}\Big(\prod_{f_a \in \mathcal{N}^{\mathcal{C}}_{i,1}} m^{(2)}_{a\to i,1}(z_{i,1}), \dots, \prod_{f_a \in \mathcal{N}^{\mathcal{C}}_{i,B}} m^{(2)}_{a\to i,B}(z_{i,B})\Big)(s_i) \tag{2.22}$$

Code Message Update:

source to factor message update:

$$m^{(2)}_{i,\omega\to a}(z_{i,\omega}) = \left[m^{(2)}_{\mathcal{G}\to i,\omega}(z_{i,\omega})\right] \prod_{f_b \in \mathcal{N}^{\mathcal{C}}_{i,\omega} \backslash f_a} m^{(2)}_{b\to i,\omega}(z_{i,\omega}) \tag{2.23}$$

where,

$$m^{(2)}_{\mathcal{G}\to i,\omega}(z_{i,\omega}) = T_{M\to 2}\Big(\prod_{s_j\in\mathcal{N}_i^{\mathcal{G}}} m^{(M)}_{j\to i}(s_i)\phi_i(s_i)\Big)_{\omega}(z_{i,\omega}) \tag{2.24}$$

and *factor to source message update:*

$$m^{(2)}_{a\to i,\omega}(z_{i,\omega}) = \sum_{\mathcal{N}_a^{\mathcal{C}}\backslash z_{i,\omega}} f_a(N_a^{\mathcal{C}}) \prod_{z_{j,\omega'}\in\mathcal{N}_a^{\mathcal{C}}\backslash z_{i,\omega}} m^{(2)}_{j,\omega'\to a}(z_{j,\omega'}) \tag{2.25}$$

and the belief of each source node is updated(in each iteration) by,

$$\hat{s}_i = \arg\max_{s_i}\Big[m^{(M)}_{\mathcal{C}\to i}(s_i)\Big]\Big[\prod_{s_j\in\mathcal{N}_i^{\mathcal{G}}} m^{(M)}_{j\to i}(s_i)\phi_i(s_i)\Big] \tag{2.26}$$

Remarks:

As in the binary case, the doping process is done by augmenting H with some additional unit weight rows to form the actual encoding matrix and the true compression rate is $r_{com} = r_{code} + r_{dope}$. However, depending on the specific configuration of the doping, for a given number of bits to dope, we may choose to dope all bits of a subset of symbols or just randomly dope the same number of bits. Discussions on the choices of doping will be provided in Chapter 4.

Chapter 3

Useful Cryptosystems

In this chapter, functionalities as well as other useful properties of the encryption schemes relevant to the thesis will be introduced.

3.1 One-Time-Pad(OTP)

In cryptography, the *one-time pad*, or OTP, is an encryption technique that is perfectly secure, as long as used appropriately. In this technique, a plaintext is paired with a random secret key. Each character in the plaintext is encrypted by combining it with the corresponding character from the key using modular addition (of their ASCII codes). It has been shown that if the key is: (i) completely randomly generated; (ii) at least as long as the plaintext; (iii) never fully or partly reused; (iv) kept completely secret, then the resulting ciphertext will be impossible to break, even if the opponent has infinite computational power. In fact, we say OTP has *perfect secrecy property.* The term *perfect* essentially means that after an opponent receives the ciphertext C, he has no more information about the plaintext P, than before receiving the ciphertext. Therefore, OTPs are also said to be *information-theoretically secure*, i.e.

$$\mathrm{H(P)} = \mathrm{H(P \mid C)}, \tag{3.1}$$

	Case 1	Case 2
Plaintext	ATTACK	GIVEUP
Key	RQBOPS	LBYKXN
Ciphertext	RJUORC	RJUORC

Table 3.1: Large Alphabet One Time Pad Encryption

where H(P) represents the entropy of the plaintext source, and H(P | C) represents the conditional entropy of the plaintext source, given the ciphertext.

The perfect secrecy of one-time pad can be clearly illustrated by the example in Table 3.1, from which we can see, having the access to the ciphertext, RJUORC, the opponent has absolutely no clues whether the plaintext was ATTACK, GIVEUP or anything else. Moreover, it is proven in [**38**] that any cipher with perfect secrecy property must use keys with effectively the same requirements as OTP keys.

However, the above stated non-trivial requirements on one-time pad for it to be perfectly secure result in many practical issues that prevent one-time pads from being widely used. For example, we have stated that the keys must be at least as long as the plaintext, and the parties conducting encryption and decryption must be able to exchange the (extremely) long key absolutely securely, which is often not practical. On the other hand, some high quality ciphers that are much easier to implement than OTPs are available and their levels of security, though not perfect, are presently deemed acceptable. Thus, these ciphers are widely used in practice. RSA and McEliece cryptosystems are some of those most frequently used schemes and we will introduce them in more depth in the following two sections.

3.2 RSA Cryptosystem

RSA[**15**] [**35**]is one of the most well known *public key cryptosystems*[**16**], where the sender of the message, say Bob, securely send messages to the receiver of the message, say Alice, in the following steps:

1. Alice generates a pair of mathematical linked key, called public key and private key, such that, the public key can be readily generated by the private key, but only with the public key, it is hard to get the private key. Public key and private key will be denoted as pbk and prk, respectively.
2. Alice transmits her public key to Bob via any (insecure) channel and keeps her private key secret.
3. Bob uses Alice's public key to encrypt the plaintext to be sent according to the agreed encryption algorithm and create a ciphertext.
4. Bob transmits the ciphertext to Alice.
5. Alice, decrypts the ciphertext using the private key that only she has access to and get the message from Bob.

Apparently, an important property of any public key crytosystem is that without the private key, the message is not decryptable. Not decryptable usually means it is computationally hard to do so. Public key cryptosystems desirably avoid the necessity of securely exchanging the same (long) key between sender and receiver, as in the case of one-time pad.

Specifically, RSA is named after Ron Rivest, Adi Shamir, Leonard Adleman. The algorithm consists of 4 major steps: key generation, key distribution, encryption, decryption. A brief introduction of each step is provided:

Key generation:

1. Select two distinct primes p and q. Empirically, to make the breaking of the cipher hard, p and q should be relatively similar in magnitude but differ in length by a few

digits.

2. Compute the modulus of both public and private keys, $n = pq$.

3. Compute $\phi(n) = \phi(p)\phi(q) = (p-1)(q-1) = n-(p+q-1)$, where ϕ is the Euler's totient function. $\phi(n)$ is kept private as part of the private key.

4. Select an integer e such that $1 < e < \phi(n)$ and $gcd(\phi(n), e) = 1$. Empirically, it is found that e should be selected such that it has short bit-length and small Hamming weight(number of 1's in its binary representation). However, it is also known that choosing an e that is too small, can be insecure in some cases. The selection of e is sometimes an art and the impact of different e's on the security can only be determined empirically. The selected e is a part of the public key.

5. Determine d, as the multiplicative inverse of e, i.e. $de = 1(mod(\phi(n)))$ and d is a part of the private key. Namely, pbk $= (n, e)$, prk $= (\phi(n), d)$. Implicitly, p and q must also be kept secret, since they can be used to calculate $\phi(n)$, which must be kept secret.

Key distribution:

(n, e) is sent to the the receiver via a reliable but not necessarily secure channel. The public keys can be reused, which is more than desirable.

Encryption:

Let M be the any arbitrary character of the message that sender wants to send to the receiver. We first need to convert M to an integer m , s.t. $0 \leq m < n$ and $gcd(m, n) = 1$. Find such m that is co-prime with n can be done efficiently with an algorithm called *padding scheme* [**38**]. After m is obtained, we compute the ciphertext as

$$c \equiv m^e \text{ (mod n)}, \tag{3.2}$$

utilizing only the public key.

Decryption:

The receiver, who has the private key, can decrypt the message by

$$m \equiv c^d \text{ (mod n)}, \tag{3.3}$$

3.3 McEliece Cryptosystem

The McEliece cryptosystem[**17**] is a public key cryptosystem whose security rests on the difficult problem of decoding an unknown error-correcting code, which is NP-hard. Such cryptosystems are called *code-based cryptosystems.*

Again, suppose Bob wants to send his message to Alice.

Key Generation:

1. Alice selects a binary (n, k)-linear error correcting code $\mathcal{C}$ that can correct up to t errors. Originally, binary Goppa codes are used for its easiness of decoding and it has resisted cryptoanalysis so far. Parameters n, k, t are agreed by the Bob and Alice beforehand. The algorithm requires the error correcting code selected must have an efficient decoding algorithm. Denote the code generating matrix, of size $k \times n$ as G.

2. Alice selects a random $k \times k$ binary non-singular matrix S.

3. Alice selects a random $n \times n$ permutation matrix P. Permutation matrix P is a square matrix that contains exactly one 1 in each row and each column. It should be seen as a linear mapping that permutes the order of the original vector, in the manner determined by the configuration of P.

4. Alice then computes the $k \times n$ matrix $\hat{G}$ such that $\hat{G} = SGP$.

Key Distribution:

Alice will send her public key $(\hat{G}, t)$ via a reliable(not necessarily secure)channel and keep her private key (S, G, P) secret.

Encryption:

1. Suppose the message that Bob wants to send Alice is $m \in Z$. Then m will first be translated into a binary sequence of length k.

2. Bob computes the vector $c' = m\hat{G}$. Note that $m\hat{G}$ in computed in the sense of modulo 2 sum.

3. Bob generates a random n-bit vector z of weight exactly t.

4. Bob computes the ciphertext as $c = c' + z$, in the sense of modulo 2 sum, and send it to Alice.

Decryption: Upon receiving c, Alice will conduct the following steps, using her private key only, to decrypt the message.

1. Computes $\hat{c} = cP^{-1}$.
2. Uses the decoding algorithm for code C to decode $\hat{c}$ to $\hat{m}$.
3. Finally, computes $m = \hat{m}S^{-1}$.

Proof of Correctness:

Since $\hat{c} = cP^{-1} = (c' + z)P^{-1} = (m\hat{G} + z)P^{-1} = mSG + zP^{-1}$, since z is chosen to have weight t and P is a permutation matrix(thus so is P^{-1}), zP^{-1} has weight at most t.

Thus the decoding algorithm, which can correct up to t errors, is able to get rid of zP^{-1} and find the corresponding $\hat{m} = mS$. Thus, original message m is thus decrypted as $m = \hat{m}S^{-1}$.

Remarks:

Though McEliece cryptosystem is computationally secure, neither it nor any code-based cryptosystem has received much acceptance in the cryptography community, partly due to the large size of public keys[**18**][**21**]. As an example, when Goppa code is used with the parameters suggested by McEliece, i.e. $k = 524, n = 1024, t = 50$, the public key $\hat{G}$ will contain $k(n-k) = 524(1024-524) = 262000$ bits, at the order of 2^{18}, which would likely cause implementation issues. The other major drawback is that the ciphertext usually has to be made much longer than the plaintext. To resolve these issues, in all simulations in the thesis, we break down the serialized binary representations of the source into shorter blocks and have encryptions done blockwisely, for better feasibility of the implementation.

In general, code-based cryptosystems are believed to be more promising in the future as storage and transmission costs go down and computation powers in machines improve. Moreover, code-based cryptosystems do have some advantages over many current practical ciphers, like RSA. The most important advantage is their (much) faster encryptions and decryptions. Take the example of McEliece cryptosystem, encryption and decryption require (only) matrix multiplications while schemes like RSA, typically require raising a number to a (large) power. Simulation results on post-encryption compression performances in Chapter 4 should give some hints on other advantages of McEliece cryptosystem in specific, over RSA.

Chapter 4

Proposed Architecture and Simulation Results

In this chapter, we will formally propose an architecture of compressing encrypted data and argue that in theory, it should enable the same compression performance as when no encryption is performed. Simulation results, however, will show this is not always the case. Some discussions on the simulation results will follow.

4.1 Source Specifics:

In this section, we will list all the sources to be compressed and give the expressions of their entropy rates. Recall that all those sources are Markov Random Fields (MRFs) and therefore we will continue using the notations in Chapter 2.

1. i.i.d. Binary Bernoulli sources:

$s = \{s_1, \ldots, s_n\}$ is an i.i.d Bernoulli source of bias p, or $\mathbf{Bern}(p)$ if $s_1, \ldots, s_n$ are independent and for $\forall i$,

$$\Pr\{s_i = \theta\} = p^{\theta}(1-p)^{1-\theta}, \tag{4.1}$$

where $\theta = \{0, 1\}$.

In the context of the pairwise factorization in (2.10), each node weight $\phi_i(\theta) = \Pr\{s_i = \theta\}$, for $\theta = \{0, 1\}$, and each the edge weight should be set to some arbitrary value, same across all edges, since there is actually no edge between any two source nodes. Apparently, the entropy, H(s), given by,

$$\mathrm{H}(s) = \mathrm{H}(p) = -p\log_2(p) - (1-p)\log_2(1-p), \tag{4.2}$$

gives us the average number of bits of information we get based on observing one symbol. It also characterizes the uncertainty, or unpredictability of the source. The higher the entropy is, the less predictable the source is. The importance of the entropy lies in its operational significance concerning coding the source, since H(s) represents the average number of bits of information per symbol from the source, we should expect that at least H(s) bits per symbol have to be used in order to represent the source without any loss of the information. This is called the **source coding theorem**, which essentially states that for any information source s:

(i). The average number of bits per symbol of any uniquely decodable code(or in the context of the thesis, the best compression rate achievable) of s must be at least H(s), the entropy of the source.

(ii). If the string of symbols is sufficiently long, there exists a uniquely decodable code for the source such that the average number of bits per symbol of the code is as close to H(s) as desired.

2. Markov Source of Order One:

As stated before, the Markov source of order 1 has the pairwise factorization. Equation(2.4) clearly shows that $\phi_1(\theta) = \Pr\{s_1 = \theta\}$. Each of all the other node potentials should be set uniform over the entire source alphabet and the edge potentials are given by the transitional densities $p(S_{i+1} \mid S_i)$. In this thesis, we will only consider the special case where $p(S_{i+1} \mid S_i)$ is the same for all i. Such Markov chains are referred to as **stationary** or **time-homogeneous**.

For a stationary Markov chain of order 1, $\mathcal{S} = S_1, \dots, S_n, \dots$, the entropy rate of the source $\mathcal{S}$ is given by:

$$\mathrm{H}(\mathcal{S}) = \lim_{n \to \infty} \mathrm{H}(S_n \mid S_{n-1}, \dots S_1) = \mathrm{H}(S_2 \mid S_1), \tag{4.3}$$

where them last step takes advantage of the fact that $\mathcal{S}$ is a Markov chain of order 1 and is stationary.

The conditional entropy $\mathrm{H}(S_2 \mid S_1)$ can then be calculated using the stationary density of each element in the alphabet as well as the transitional densities as follows:

without loss of generality, let the alphabet of the source be $\{1, ...m\}$ and let μ_i denote the steady state probability of i, and let p_{ij} denote the conditional probability $\Pr\{s_{k+1} = j \mid s_k = i\}$, for all k, then the source entropy $\mathrm{H}(s)$ is given by:

$$\mathrm{H}(s) = -\sum_{i=1}^{m} \mu_i \sum_{j=1}^{m} p_{ij} \log_2(p_{ij}), \tag{4.4}$$

Note that:
(i) The convention $0\log_2(0) = 0$ is used.
(ii) Given the state transition matrix P, with $\mathrm{P}_{ij} = p_{ij}$, $\forall i, j \in \{1, \dots, m\}$, the stationary densities $\mu = (\mu_1, \dots \mu_m)$ can be found. More specifically, when the Markov chain is in its steady state, the stationary density of any arbitrary state j, should satisfy: $\mu_j = \sum_i \mu_i \mathrm{P}_{ij}$, or compactly, $\mu = \mu\mathrm{P}$, from which we can solve for the stationary densities μ.

3. 2D Binary Ising Model:

Recall equation (2.6), here we only consider the *homogeneous* and *symmetric Ising model*, where $\phi_i(1) = 1 - \phi_i(0) = p = 1/2$ for all i and $\varphi_{ij}(0,0) = \varphi_{ij}(1,1) = 1 - \varphi_{ij}(0,1) = 1 - \varphi_{ij}(1,0) = 1 - q$, for $\forall(i,j) \in \mathcal{E}$. We refer to this source as **Ising**(0.5, q).

The entropy of 2D **Ising**(0.5,q) source is much more involved. Onsager gives an exact expression in **[19]** of the entropy of Ising source, with the limit of the number of nodes going to infinity. The results will be given here with the proof omitted. Interested readers are referred to **[37]** and **[19]** for more details. Let q denote the probability of neighbouring nodes taking different values, and let N be the number of nodes in each row and each column. Also, assume periodic boundary conditions, then the entropy of 2D **Ising**(0.5, q) , $H(s)$ is given by:

$$H(s) = \lim_{N\to\infty} N^2 h(s), \tag{4.5}$$

where $h(s)$, called the entropy density, is given by(with respect to ln, instead of log_2):

$$\begin{aligned} h(s) = \ln(2) + \frac{1}{8\pi^2}\int_0^{2\pi}\int_0^{2\pi} & \ln[a^2 - b(cos(\theta_1) + cos(\theta_2))]d\theta_1 d\theta_2 \\ & - \frac{\mathrm{J}}{4\pi^2}\int_0^{2\pi}\int_0^{2\pi} \frac{2ab - a(cos\theta_1 + cos\theta_2)}{a^2 - b(cos\theta_1 + cos\theta_2)} d\theta_1 d\theta_2, \end{aligned} \tag{4.6}$$

where

$$a = \frac{2q^2 - 2q + 1}{2q - 2q^2} \tag{4.7}$$

$$b = \frac{1 - 2q}{2q - 2q^2} \tag{4.8}$$

$$\mathrm{J} = \frac{1}{2}\ln(\frac{1-q}{q}) \tag{4.9}$$

We see that, once parameters a, b, J are determined for a given q, entropy density may be calculated numerically. Finally, in order to convert the entropy to be in bits (with respect to $\log_2$), we need to divide the entropy with respect to ln by the factor of ln2.

In practice, when the size of the grid is relatively small, the gap between the true entropy of the source and the limiting entropy can be significant. Therefore, Markov Chain Monte Carlo(MCMC) simulation techniques[**20**] are used to give approximations of the true source entropy. This approximated entropy is used as a more important benchmark for the compression performances. The following figure gives the limiting entropy as well as the approximated entropies for **Ising**(0.5,q) sources of lengths 100, (approximately) 1000, 10000, respectively:

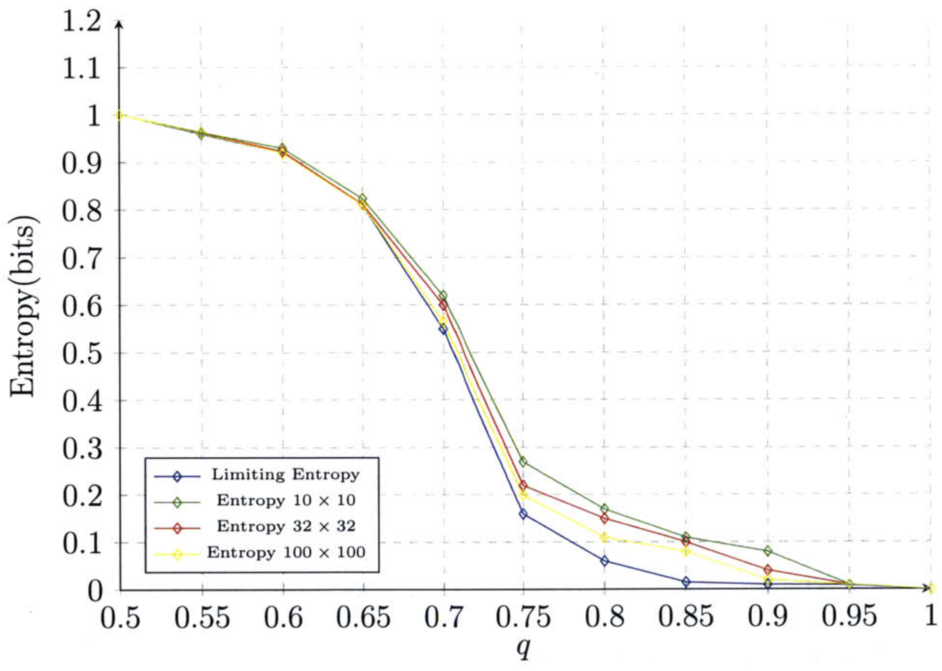

Figure 4-1: Entropy of Ising models of different sizes

4.2 Compression of Encrypted Data:

In this section, we will first propose the algorithm of compressing the encrypted data and then argue that the theoretical performance of compressing an encrypted source should be the same as that when no encryption is ever performed(in subsection 4.2.1). In subsections 4.2.2 and 4.2.3, respectively, simulation results on both synthetic and real world data will be given, followed by discussions on various aspects of the algorithm.

4.2.1 An algorithm of compressing encrypted data and its performance in theory:

Let $s = \{s_1, \dots, s_n\}$ be the MRF source with joint distribution $p(s)$. Moreover, let k be the keyword and the source be encrypted to ciphertext $k(s)$. Generally speaking, some number of consecutive source symbols, say l, can form up a block and be encrypted jointly, as if a larger character were encrypted and a larger alphabet were used. Here, for demonstrative purposes, we will proceed as the source is encrypted symbolwisely. In fact, it will become clear later under "on Symbolwise Encryption vs Blockwise Encryption"(pg. 70-71), an overly large alphabet will significantly increase the computational complexity of the algorithm. So a small l, for example $l = 1$, corresponding to symbolwise encryption, is desirable. With $l = 1$, we will use $k(s)_i$ to denote the ciphertext for source node s_i. Recall that, in section 2.3.1, we introduced the compression architecture in which the source model needs to be made available **only** at the decoder. Imagine, we now have both the source model $p(s)$ and the key k available at the decoder, then a new symbol translation pair, (with liberties taken in notations), $t'^{(k)}_{M\to 2}$ as well as $t'^{(k)}_{2\to M}$, can be defined such that a symbol and the binary representation of its ciphertext can be converted from one to another. Note that, the translation pair is a function of the key k, and thus k is used to parameterize the translation pairs. Accordingly, we can define the message translation pair: $T'^{(k)}_{M\to 2} : (\mathbf{S} \to \mathbf{R}^+) \to (\mathbf{GF}(2) \to \mathbf{R}^+)^{B'}$ and $T'^{(k)}_{2\to M} : (\mathbf{GF}(2) \to \mathbf{R}^+)^{B'} \to (\mathbf{S} \to \mathbf{R}^+)$ that converts between a message $m^{(M)}$ over $\mathcal{S}$ and a B'-tuple of messages $m^{(2)} = m_1^{(2)}, \dots m_{B'}^{(2)}$ over $\mathbf{GF}(2)$, where B' is the bit-length of each symbol's ciphertext. Therefore, in theory, our model-free compression structure, should see the problem of compressing encrypted source data merely as an ordinary model-free compression problem, but with a different message translation pair defined, so that the encoding and decoding(joint decompressing and decryption) algorithms are essentially the same as the ones described in section 2.3.

Essentially, at the decoder, during each iteration, the message sent to any source symbol s_i from the code subgraph, $m^{(M)}_{\mathcal{C}\to i}(s_i)$ is obtained by message decryption followed by conventional translation $T_{2\to M}$. The message sent from the graph subgraph to each bit of the source nodes' representational map, $m^{(2)}_{\mathcal{G}\to i,\omega}(z_{i,\omega})$, is obtained by conventional message translation $T_{M\to 2}$ followed by message encryption. The rest of the decoding algorithm remains unchanged.

Moreover, since the encryption scheme must be a one to one mapping to qualify as a valid encryption scheme, we have $\mathrm{H}(s) = \mathrm{H}(k(s))$ (for any given key k), from which we see the limit of the compression performances should not be affected. Therefore, in theory, the proposed algorithm should be able to compress an encrypted source as well as it compresses the original, unencrypted source.

4.2.2 Simulation Results on Synthetic Data

In the following subsection, we will show the simulation results of compressing encrypted sources. The results will show that, with the only exception of compressing one-time-pad encrypted data, there is always performance loss in compression when an encryption is performed prior to the compression, compared to when no encryption is performed. Note that, the source data in this subsection is **synthetic data**, which means it is generated exactly according to the underlying model so there is no model-data mismatch that would necessarily result in performance loss in compression.

For data sources, we consider binary sources and large-alphabet sources. We will encrypt binary sources(binary Bernoulli i.i.d. sources, binary Markov chain of order one, binary 2D Ising models, etc.) only with bitwise OTP and encrypt large-alphabet sources with schemes that are more naturally defined over integers, i.e. large-alphabet OTP, McEliece cryptosystems, RSA. Note that, when dealing with binary Bernoulli and Markov sources, we could choose to treat it as large alphabet sources by binding some number of bits, say b, together as an integer and encrypt each integer with integer-based encryption schemes(e.g. McEliece, RSA, etc.). However, this is not really necessary since if a binary source is bitwise Bernoulli i.i.d. (resp. bitwise Markov source of order 1) then the binded large-alphabet source is also symbol-wise i.i.d(resp. symbol-wise Markov of order 1). Therefore, the above stated attempts will be incorporated later when we encrypt large-alphabet sources with integer-based encryption schemes.

In each of the following simulations, let B be the bit-length of each source symbol and n be the number of symbols in the source vector. Then for each given encoding matrix H of size $k \times nB$, its compression rate r_{com} is defined as $r_{com} = r_{code} + r_{dope}$, where the coding rate, r_{code} = rank(H)/nB(which usually can be made to equal $r_{nom} = k/n$B by selecting H such that rows are independent), and the doping rate r_{dope} = (number of bits doped)/nB. Furthermore, we define the best achievable compression performance, or the **best compression performance** as the lowest compression rate r_{com}, such that the encrypted source can be recovered losslessly(the algorithm converges to the correct result in 500 iterations). Given H, the doping rate, as well as which bits to dope, are tuning parameters that we can work on to optimize the best compression performance. For each simulation result below, the doping rate used will be specified. In the plots below, the best compression performances versus (essentially) different source entropies, are provided. Unless otherwise specified, each data point is obtained as the average over 20 trials.

(1)Binary Bernoulli i.i.d. Sources, Encrypted by OTP

Figure 4-2 describes the performance of compressing (OTP-encrypted and unencrypted) binary Bernoulli i.i.d sources of different values of p=Pr$\{s_i = 0\}$, $\forall i$. The source length $n = 1000$ and $r_{dope} = 0$.

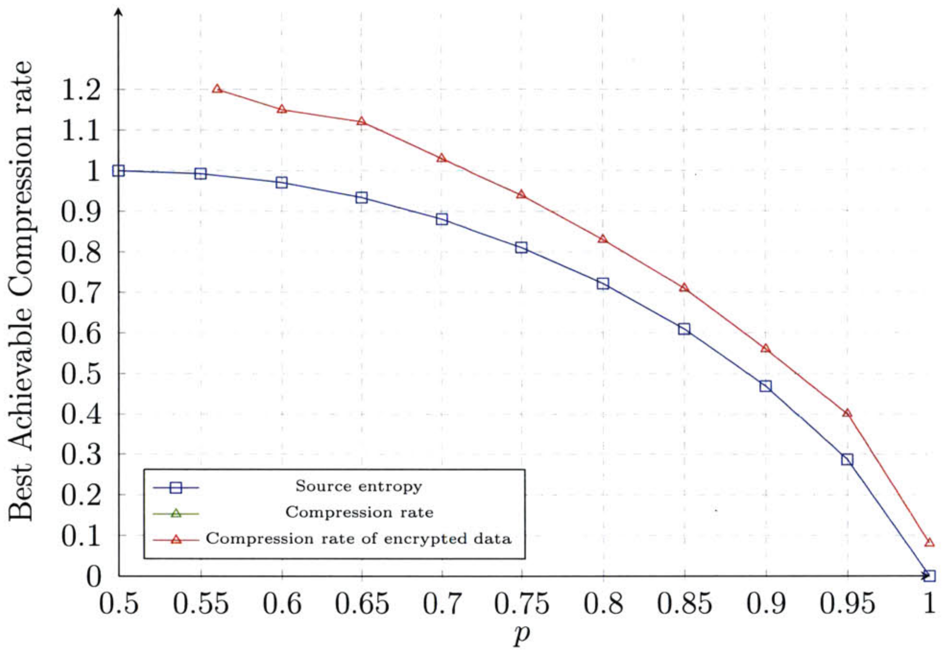

Figure 4-2: Compression of OTP-Encrypted Binary Bernoulli Sources

Observations:

(a). The performance of compressing encrypted data is as good as that when no encryption is performed.

(b). We also tried $r_{dope} = 0.01$ as well as $r_{dope} = 0.1$ but did not see visible differences in their best compression performances(and thus those curves are omitted) and the impact of different doping rates seems to determine only the rates of convergence–the larger the doping rate is, the faster the algorithm converges.

(c). There is a gap between the best achievable compression rates and the entropy rates. The gap is much larger in the high entropy(or high rate) regions than in the low entropy(or low rate) regions.

(2)Binary Markov Source, Encrypted by OTP

In Figure 4-3(doping rate=0) and Figure 4-4(doping rate=0.1), the horizontal axis denotes $p_{stay} = \phi_i(0,0) = \phi_i(1,1) = \Pr\{s_i = \theta \mid s_{i-1} = \theta\}$, $\theta = \{0,1\}$, $\forall i$, in a homogeneous binary Markov chain of order 1. For each doping rate, we tested source lengths $n = 1000$ and $n = 10000$.

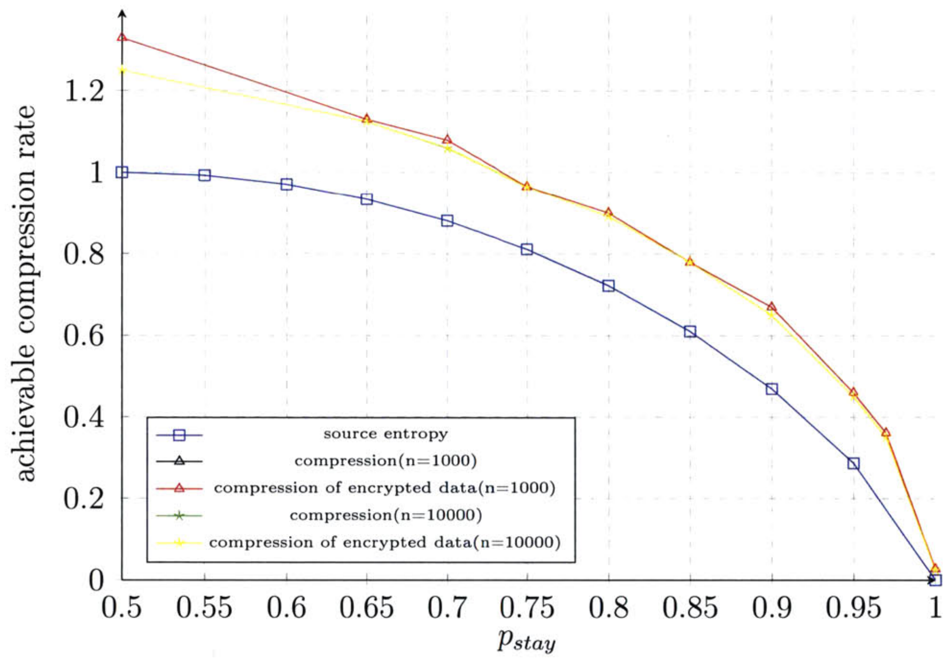

Figure 4-3: Compression of OTP encrypted Binary Markov source, doping rate=0

Observations:

(a) For each doping rate and each source length, the performance of compressing OTP-encrypted sources is the same the as that of compressing the same source without encryption.

(b) For both doping rates, compression performances on the source of length $n =$ 10000 are better than those when $n = 1000$. However, the differences are not significant. We conjecture the performances of $n = 10000$ slightly outperform those of $n = 1000$ since for a given θ, a longer portion of the Markov chain was in its station-

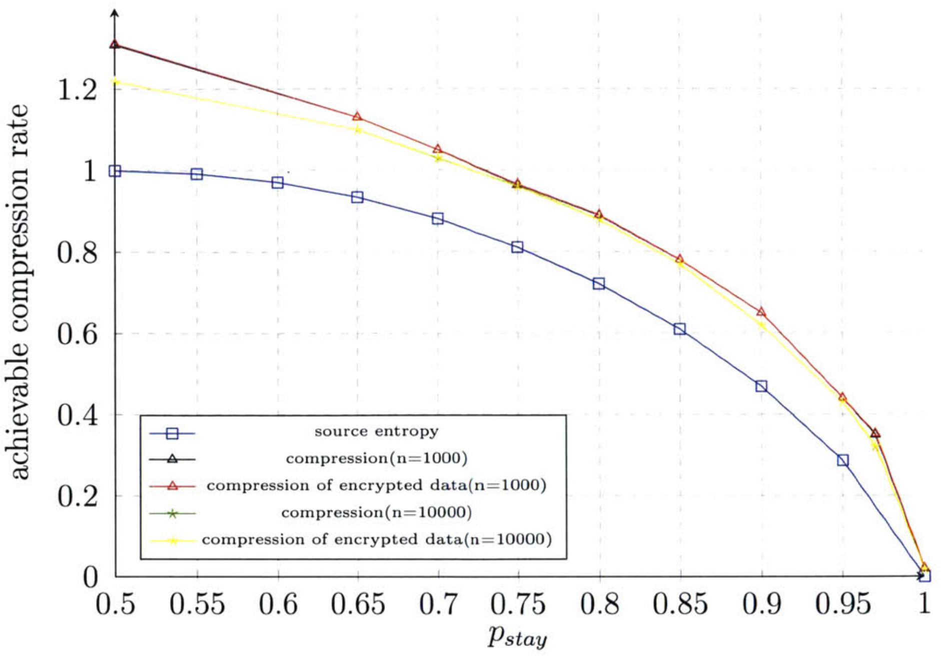

Figure 4-4: Compression of OTP encrypted Binary Markov source, doping rate=0.1

ary distribution when $n = 10000$. However, for all θ's, it takes much fewer random variables than even $n = 1000$ for two-state Markov chains to reach the stationary distribution. Therefore, the differences in performances are expected to be small, which is also found from the simulations.

(c) $r_{dope} = 0.1$ enables better performances than $r_{dope} = 0$, although the improvement is small relative to the gap between the best compression performances and the entrópy rates. Also, the proposed architecture again demonstrates its competitiveness in low entropy regions, i.e. the region with large θ's.

(3)Binary Ising Model, Encrypted by OTP

Figures 4-5 and 4-6 show the best achievable compression rates of OTP-encrypted **Ising**(0.5,q) sources of different sizes(10×10 and 100×100, respectively). The best compression rates obtained under different doping rates are all compared with the limiting entropy rates as well as the simulated source entropy of the corresponding size. In all cases, best compression rates are the average of the best compression rates of 50 trials.

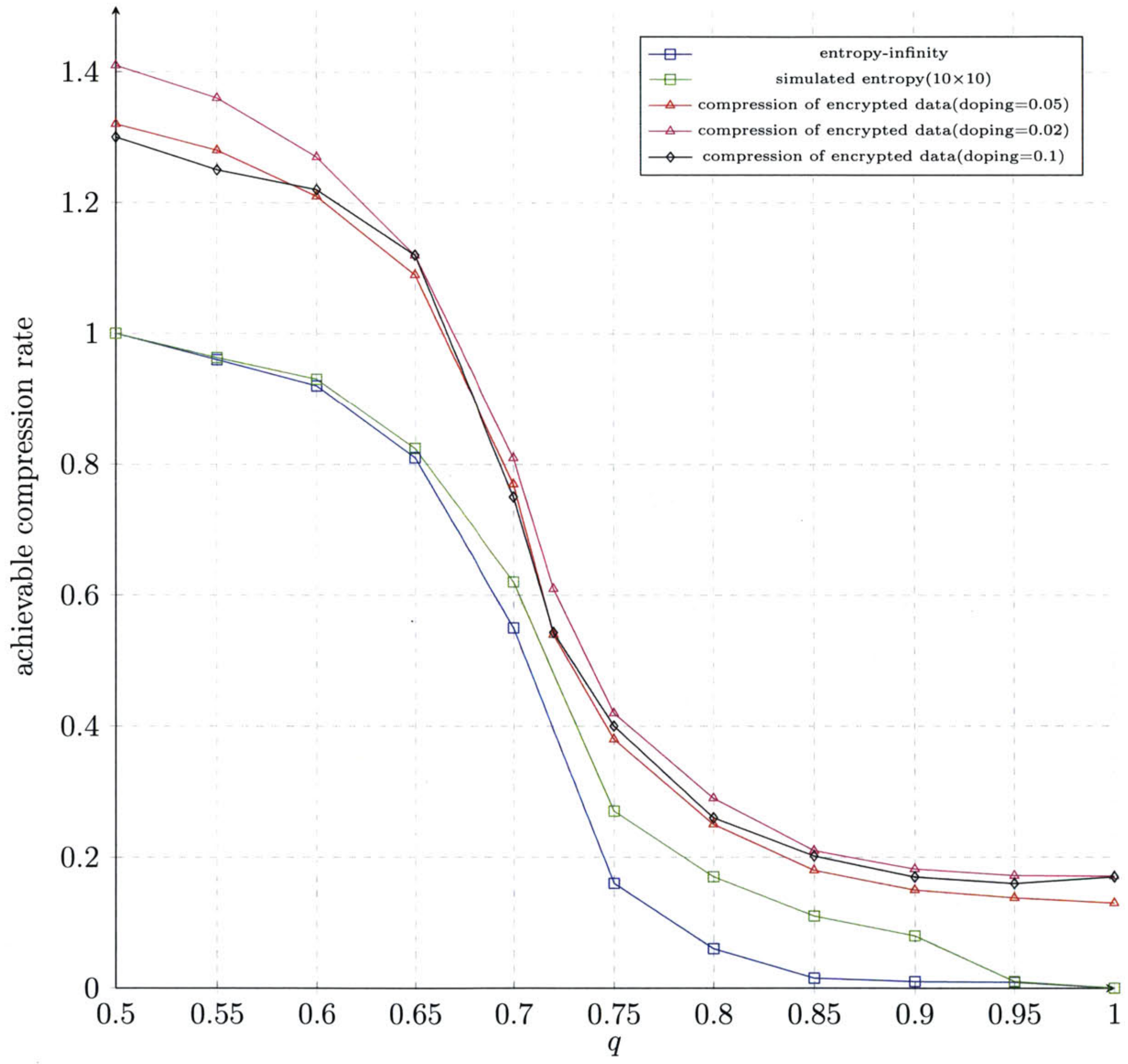

Figure 4-5: Compressing OTP-Encrypted 10×10 Ising Model

Observations:

(a) For both source sizes and all doping rates, compression performances are not jeopardized by OTP (thus for simplicity, curves of compression of the original, unencrypted sources are omitted in Figure 4-5 and Figure 4-6).

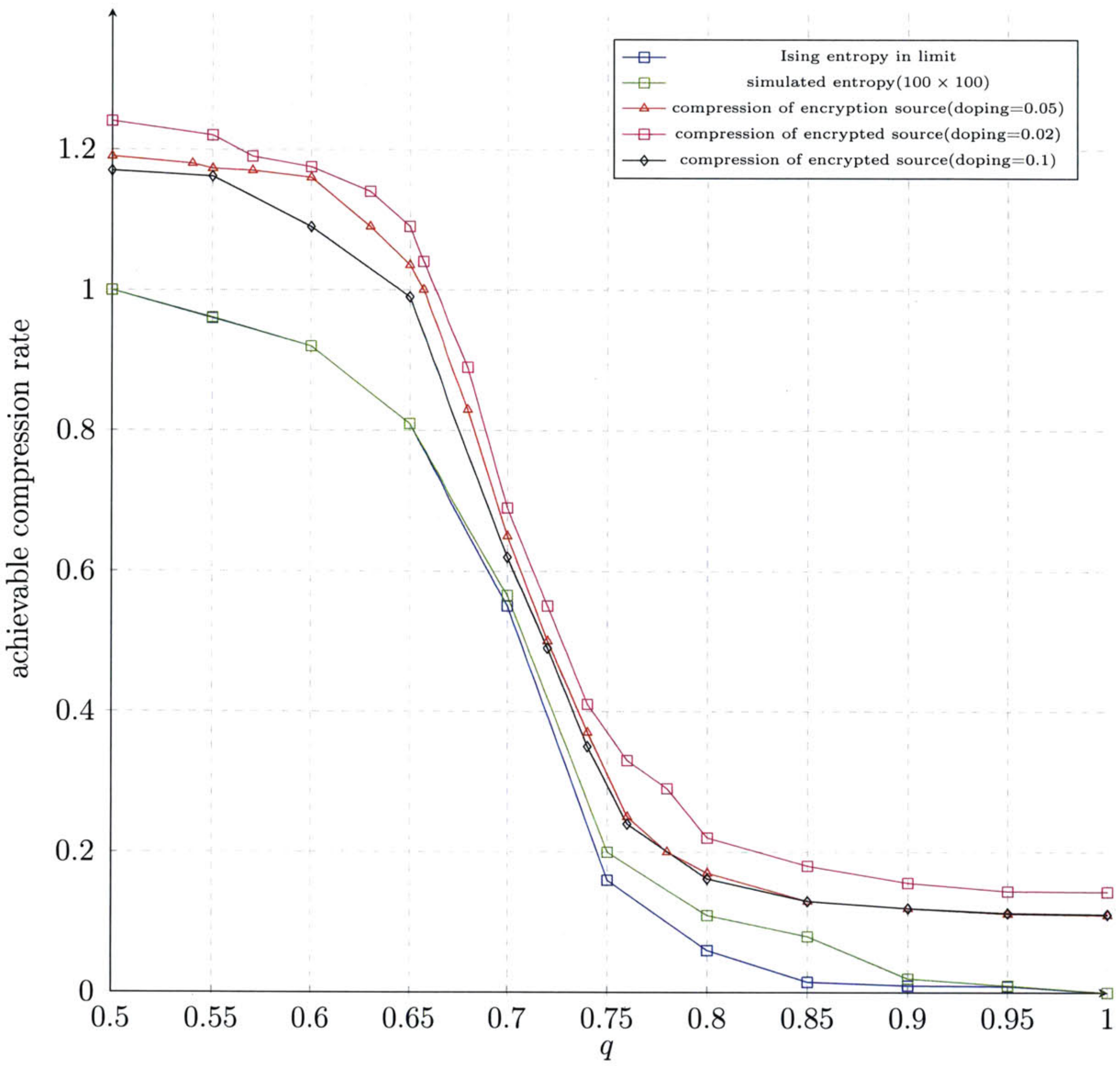

Figure 4-6: Compressing OTP-Encrypted 100 × 100 Ising Model

(b) For both source sizes, $r_{dope} = 0.05$ enables better compression performances(of OTP encrypted data) than $r_{dope} = 0.02$. However, $r_{dope} = 0.1$ enables better performances (for most entropy rates) than $r_{dope} = 0.05$ only for the 100×100 Ising source and the performance gap is more significant at high entropy region. On the other hand, for the case of the 10×10 Ising source, compression performances under $r_{dope} = 0.1$ are worse than those under $r_{dope} = 0.05$, except for in the very high entropy region, where a higher doping rate is needed intuitively. Based on the above facts, we will refer to $r_{dope} = 0.02$ as an **insufficient doping rate** for both 10×10 and 100×100 Ising sources, where an appropriate increase of the doping rate will help the algorithm converge correctly and such positive effect outweighs the negative effect that a higher r_{dope} is added in when calculating r_{com}. Similarly, $r_{dope} = 0.05$ is insufficient for the 100×100 Ising source. However, for the 10×10 Ising source, from the fact that even $r_{dope} = 0.055$ results in slightly worse compression performances than $r_{dope} = 0.05$ does(the curve corresponding to $r_{dope} = 0.055$ was not shown in Figure 4-5 since it is rather close to that of $r_{code} = 0.05$), we conclude that $r_{code} = 0.05$ is **sufficient** for the 10×10 Ising source – the positive effect that an even higher doping rate has on the algorithm converging correctly is outweighed by the negative effect of a higher r_{dope} being added, when computing r_{com} numerically. Therefore, if we want to find the optimal doping rate for the 10×10 Ising source, we could start with $r_{dope} = 0.05$ and gradually decrease the doping rate until it becomes insufficient again. Likewise, we may find the optimal doping rate for the 100×100 Ising source by gradually increasing the doping rate from 0.05 until it becomes sufficient.

(c) For a given doping rate, the 100×100 source has better compression performances than the 10×10 source. Performance difference is larger in the high rate region than in the low rate region.

(d) Also, it is worth noticing that there is a phase transition around $q = \sqrt{2}/2$, such that long-range correlations are high above this threshold[**19**][**37**]. Significant compression gains are achievable above this threshold, in the low rate region. However, in those cases, the Ising source is in the ferromagnetic phase, and the Gibbs samples of the source are likely to be trapped in a state with high probability and the Gibbs sampling can take very long to reach other states. Therefore, the average (best achievable) performance over 50 trials for each of those high q cases, likely represents only the compression performances with respect to realizations of several highly probable states.

Some General Remarks on OTP:

Before investigating the case of compressing encrypted large-alphabet sources, we make some general remarks on the effect of OTP on post-encryption compression performances. In all cases above, we see that OTP does not jeopardize the compression performances. This is just as expected– given the key, the encrypted source is just another source of the same statistical structure, same length and same entropy, therefore the algorithm merely sees a compressing encrypted source problem as a compressing unencrypted source problem and the performance of compressing an OTP(including generalized OTP) encrypted source is thus the same as compressing the same source without encryption. Recall the description of Chapter 3, the main problem with OTP is that it is not practical– the length of the key has to have the same length as the source and be purely randomly generated, which is not feasible in most real applications – therefore, in all simulations to follow, we won't explicitly show performance curves of OTP, but will instead focus on more practical schemes, e.g. McEliece Cryptosystem and RSA.

(4). Large-alphabet Markov Source of order 1, Encrypted by RSA and McEliece Cryptosystem

We now consider homogeneous irreducible aperiodic Markov chain of length $n = 1000$, over GF(M), where M>2 is the size of the source alphabet. Namely, source $s^n = \{s_1, \ldots, s_n\}$, with each element taking a value in alphabet $\mathcal{A} = \{0, 1, \ldots, (\mathrm{M}-1)\}$ and **transitional densities, or second order statistics** $\Pr\{s_{k+1} = \beta \mid s_k = \alpha\} = p_{\alpha\beta}$, $\forall k = 1, \ldots (n-1)$, where $\alpha, \beta \in \mathcal{A}$. We here consider cases $\mathrm{M} = 16$(a.k.a 4-bit sources) and $\mathrm{M} = 256$(a.k.a. 8-bit sources).

The expression of source entropy $\mathrm{H}(s)$ was given in section 4.1 and one of its variants, *bitwise entropy*, $\mathrm{H}_b(s)$ is defined as, $\mathrm{H}_b(s) = \mathrm{H}(s)/\lceil \log_2 \mathrm{M} \rceil$. Bitwise entropy is particularly useful in our application since we can hope to compare compression performances over sources with alphabets of different sizes.

In simulations below, 4-bit and 8-bit Markov sources of length $n = 1000$ with different bitwise entropies are symbolwisely encrypted by RSA and McEliece cryptosystems, respectively, and the best compression rates are compared to those without encryption as well as the bitwise entropy of the source. In the descriptions of observations to follow, "McEliece(resp. RSA) performance" of a certain source, refers to the compression performance of the source encrypted by McEliece(resp.RSA) and "no encryption performances" refers to the performances of compressing the unencrypted sources.

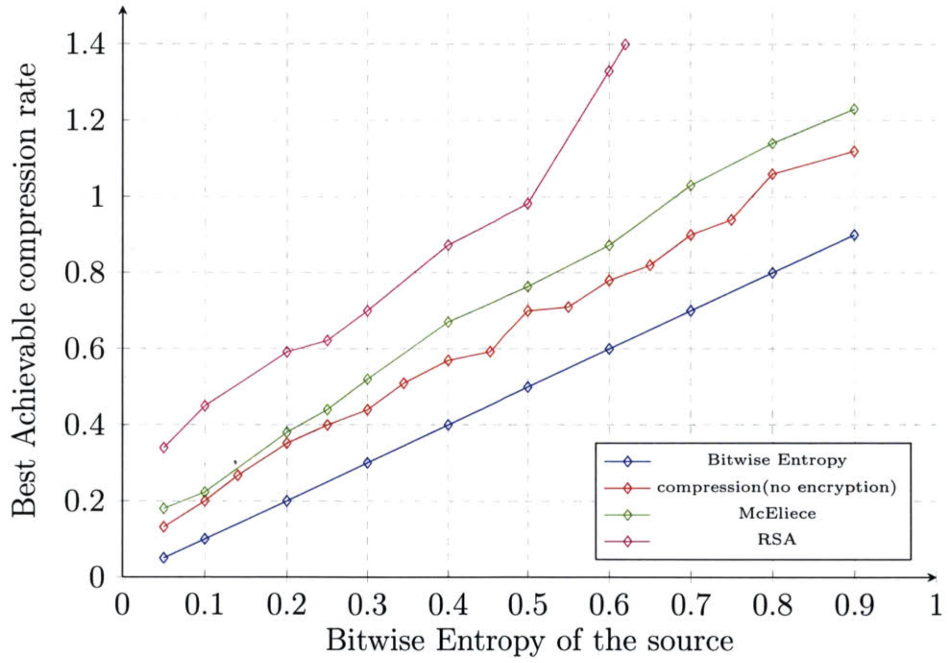

Figure 4-7: Compressing encrypted 4-bit Markov sources,doping=0.05

Observations:

(1). There is a visible gap between the McEliece performances and the no-encryption performances. The gap is relatively small, compared to the inherent gap between the no-encrypton performances and the source entropy, in the low rate region and gets larger as entropy gets large. The gap between RSA performances and no-encryption

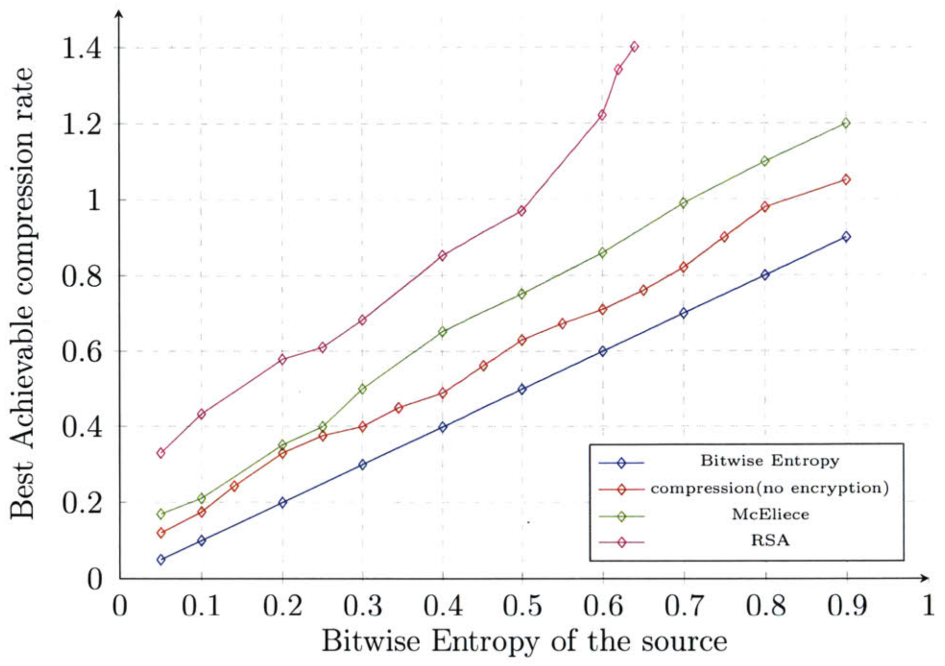

Figure 4-8: Compressing encrypted 4-bit Markov sources,doping=0.10

performances is larger than that between McEliece performances and no-encryption performances.

(2). Source entropy and doping rate kept same, the no-encryption, McEliece and RSA performances are slightly better for 8-bit sources than their counterparts for 4-bit sources.

(3). All other factors kept equal, $r_{dope} = 0.1$ enables better compression performances than $r_{dope} = 0.05$ for both 8-bit and 4-bit sources. Moreover, further investigations show that $r_{dope} = 0.1$ is close to(all rates above $r_{dope} = 0.12$ have been found empirically to be sufficient for both 8-bit and 4-bit sources) a sufficient doping rate for both 8-bit and 4-bit sources.

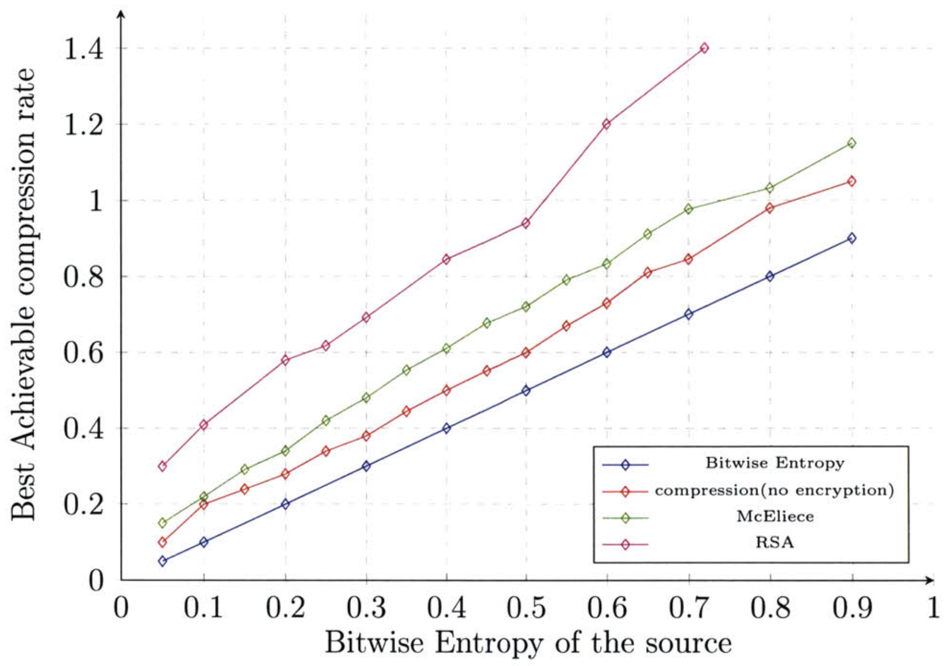

Figure 4-9: Compressing encrypted 8-bit Markov sources,doping=0.05

4.2.3 Compressing Encrypted text files

In all simulations above, we have limited our attentions to synthetic data, where there is no mismatch between the data and the underlying source model. Strictly speaking, the Gibbs samples of the Ising model are only (very good) approximates, but they can be treated as exact in this application. However, data from the real world usually do not come from such simple source models and therefore it's usually hard to implement the above algorithm with the exact model underlying the observed data. Therefore, when using some simple source models to approximate the exact model, some performance loss due to the model-data mismatch is expected. However, such approximation is still desirable since it brings about the feasibility of some fundamental operations(compression, inference) that essentially relies on the simpleness of source models.

One of the initial motivations of this research was to move toward practical compression of encrypted English text files. We will take in as inputs English files excerpted to some fixed length that guarantees the computational feasibility. Using the language of

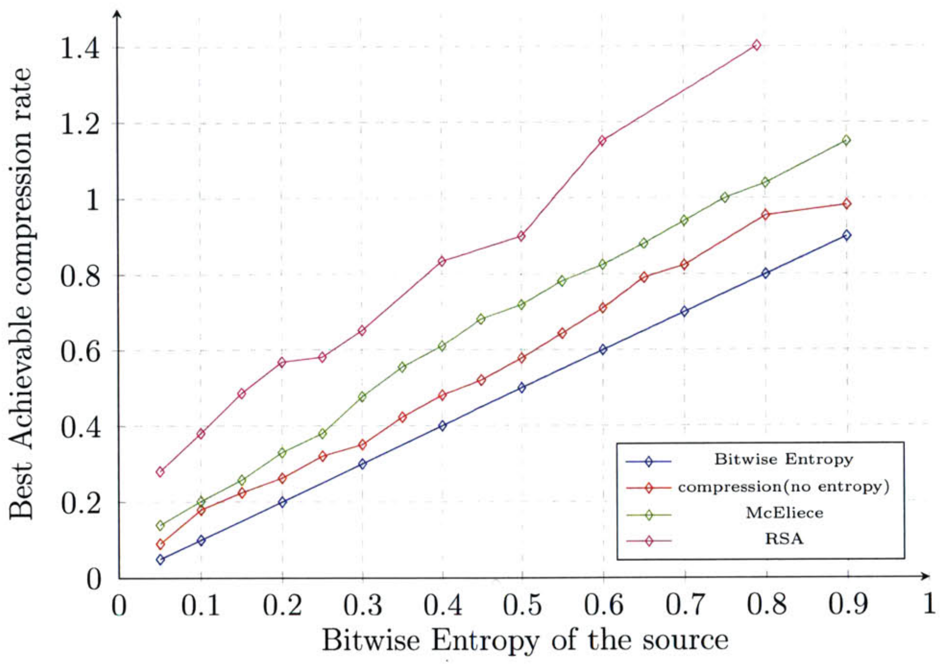

Figure 4-10: Compressing encrypted 8-bit Markov sources,doping=0.10

the previous chapters, the potentials(node potentials, edge potentials, etc.) necessary for the algorithm are taken from the publicly available zeroth, first and second-order models for English[**22**]. Entropies derived from the first and second order models are 3.36 bits/character and 2.77 bits/character, respectively. In limit, Shannon has estimated the entropy rate of the 27-letter English text to be 2.3 bits/character[**26**]. As stated, compression performances are expected to be worse than those of the synthetic data cases, since the Markov models of different orders are all approximations of the English source. The higher the order of the Markov model is, the better the approximation is, while at the same time, the higher the algorithm's complexity is. There is always a trade-off.

Now we describe the simulations to be conducted:

We categorize files to be encrypted-and-compressed into Group A and Group B. Group A contains two files [**23**] [**24**] that are among the classical literatures based on which the first and second order Markov models of English are developed. Group B contains 10 random articles, chosen from the Oct.10th, 2015 *Economist* magazine.

Each file in each group is excerpted to have length(in characters) n=10000 and is modified so that the cardinality of the alphabet=27(only lowercase letters as well as the space are considered). Thus, the bit length of the source is 50000. The choice of bit length aims at guaranteeing reasonable long file length while maintaining algorithm's computational feasibility.

Each file in each group is encrypted by McEliece and RSA, i.e. each file now has two encrypted versions. Then, two best compression performances are obtained for each encrypted version of each file: one assumes, at the decoder, that the source has the first order Markov model and the other assumes that the source has the second order Markov model. Note that, slight changes from equations (2.21) to (2.26) are needed when second order Markov model is assumed. Namely, we need to modify those equations to incorporate the potentials defined over cliques of length 3. The performances of each test(no encryption, McEliece, RSA) for the classical files(files 1 and 2)are reported in Table 4.1 and Table 4.2. For simplicity, for articles in *Economist*, only the *average* of the best compression rates, over the 10 articles, is reported in Table 4.3.

Moreover, we also generated two synthetic files according to the first and second order Markov models of English, respectively. The same tests are conducted, with correct source model assumed. Results are reported in Table 4.4. In all simulations mentioned above, $r_{dope} = 0.1$ is used. All numbers in the tables are in the unit of bits.

In each of the four tables, we can see the effects of different encryption schemes on post-encryption compression performances by comparing post-encryption compression performances corresponding to a certain encryption scheme with the "No encryption performances". The row led by "1st order" (resp."2nd order") provides information on the presumed bitwise source entropy rate as well as the best-compression performances of different tests, when the first order(resp. second order) Markov model is assumed by the decoder.

Observations on the simulation results:

File 1	presumed bitwise entropy rate	No encryption	McEliece	RSA
1st order	0.672	0.922	>1	>1
2nd order	0.554	0.72	0.838	>1

Table 4.1: Compression of Classical File 1

File 2	presumed bitwise entropy rate	No encryption	McEliece	RSA
1st order	0.672	0.918	>1	>1
2nd order	0.554	0.742	0.855	>1

Table 4.2: Compression of Classical File 2

(a) The proposed architecture is not the most competitive on compressing English files, which can be inferred from previous simulations on synthetic data, where we found that the proposed architecture works particularly well in the low rate(low entropy) region, but not that well in the high rate(high entropy) region. However, according to the first (resp. second) order Markov models of English, the bitwise entropy of the source is 0.672(resp. 0.554) bits, both of which are in the (medium-)high rate region.

(b) From the best no encryption compression performances, there seems to be smaller data-model mismatch in classical files than in Economist files, which is expected since the classical files(File 1 and File 2) are among the ones from which the first and second order Markov models of English are estimated. Moreover, from the fact that in each table, a smaller gap between the compression performances and the source entropy is achieved when second order Markov model is assumed at the decoder, than when first order model is assumed, it is also verified that the second-order model is a better approximate of the 27-letter English source than the first-order model.

(c) Same as the synthetic data case, McEliece cryptosystem, outperforms RSA in real world data sources. We will discuss why they result in very different post-encryption compression performances in section 4.3.1 and further in Chapter 5.

Economist	presumed bitwise entropy rate	No encryption	McEliece	RSA
1st order	0.672	0.969	>1	>1
2nd order	0.554	0.773	0.892	>1

Table 4.3: Compression of Encrypted Economist files

Synthetic Data	bitwise entropy rate	No encryption	McEliece	RSA
1st order	0.672	0.848	0.962	>1
2nd order	0.554	0.702	0.81	>1

Table 4.4: Compression of Encrypted Synthetic Markov Data for English

4.3 Discussions

4.3.1 Discussions on the results

From simulation results above, we see that the post-encryption compression performances of McEliece and RSA were visibly worse than no-encryption compression performances, with the former to be relatively better. Since McEliece and RSA are practical encryption schemes, the results indicate that the McEliece cryptosystem could be promising for our applications, at least in the low rate regions. We will use this section to discuss why McEliece cryptosystems can outperform RSA in the aforementioned application.

On Comparing RSA and McEliece Cryptosystems

Recall that we have stated earlier in section 2.3 that when dealing with large-alphabet sources, the proposed algorithm of compressing encrypted data will incur no architectural loss at the message translation layer if and only if bits in a symbol's binary representation(in this specific application, bits in the binary representation of a symbol's ciphertext) are independent, if each bit in symbols' binary representations is seen as as a random variable, given the key for a certain encryption scheme. However, the independence condition is often not satisfied. In fact, as long as the encryption scheme encrypts a symbol to a longer integer(in the sense of bit length), then the resulting binary representation of a symbol's ciphertext cannot contain completely independent bits. Therefore, to understand the role of a certain encryption scheme in the post-encryption compression performances, we should investigate how close the resulting binary representation of a symbol's ciphertext is from an independent binary sequence. We therefore design the following experiment:

Let alphabet $\mathcal{A} = \{0, 1, \ldots, 15\}$, with alphabet size M=16 and bit-length of the source B=4. We assume equal priors among all symbols in the alphabet, i.e. $\pi_0 = \pi_1 = \ldots = \pi_{15} = \frac{1}{16}$, so that we can see that the four bits in the binary representations of symbols are independent and each has equal marginals, i.e. $p_i(0) = p_i(1) = 0.5, \forall i = \{1, 2, 3, 4\}$, where i refers to the index of the bit counting from the left in source symbols' binary representations. Now, we encrypt each character in the alphabet by McEliece cryptosystem and RSA, respectively, in the following manner:

McEliece Cryptosystem:

For a simpler demonstration, we will here use the McEliece cryptosystem based on (7,4) Hamming code. Using notations in previous chapters:

$$G=\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix},$$

$$S=\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$P=\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix},$$

where we can see that S is invertible; G is the code generator matrix with a known, fast decoding algorithm of a error-correcting code that can correct $t = 1$ error; P is a permutation matrix. Therefore (S, G, P) jointly form up a valid private key.

Thus, the public key $\hat{G}=\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix},$

Moreover, we randomly choose a weight 1 noise vector $z = (1, 0, 0, 0, 0, 0, 0)$. Therefore, each plaintext word $m \in \{0, 1\}^4$ will be encrypted as $c = m\hat{G} + z$, where all sums involved are in the sense of modulo 2. Table 4.5 lists all words of length 4 and their ciphertexts.

We can see, given all 16 symbols in the plaintexts having uniform priors, each of the seven bits in the binary translation for ciphertexts has equal densities for 0 and 1. Moreover each combination of two, three or four bits are still independent. The proof is trivial and is omitted here.

We also have done several similar simulations using other valid priviate keys (S, G, P) and z and find the above property to be true in all cases encountered. Though rigorous proof that generalizes to all valid (S, G, P) and z may not be simple, we conjecture that the above property can be generalized to general McEliece cryptosystems based

character	binary translation	binary translation for ciphertexts
0	0 0 0 0	1 0 0 0 0 0 0
1	0 0 0 1	1 0 0 1 1 1 1
2	0 0 1 0	1 1 0 1 0 1 0
3	0 0 1 1	1 1 0 0 1 0 1
4	0 1 0 0	1 0 1 1 1 0 0
5	0 1 0 1	1 0 1 0 0 1 1
6	0 1 1 0	1 1 1 0 1 1 0
7	0 1 1 1	1 1 1 1 0 0 1
8	1 0 0 0	0 0 0 0 1 1 0
9	1 0 0 1	0 0 0 1 0 0 1
10	1 0 1 0	0 1 0 1 1 0 0
11	1 0 1 1	0 1 0 0 0 1 1
12	1 1 0 0	0 0 1 1 0 1 0
13	1 1 0 1	0 0 1 0 1 0 1
14	1 1 1 0	0 1 1 0 0 0 0
15	1 1 1 1	0 1 1 1 1 1 1

Table 4.5: (4,7) McEliece Encryption

on Hamming codes.

RSA:

Now, let's see a comparative example of RSA. The choices of parameters in the following example are all valid according to Chapter 3, though they may not be practically desirable for the considerations of security due to their small sizes– they are used here only for demonstrative purposes. We will use RSA to encrypt four-bit symbols to seven-bit symbols, same as the case of McEliece cryptosystem, and see how close to an independent sequence the encrypted binary sequence is. From there, we may discover why the McEliece cryptosystem may have outperformed the RSA in post-encrypting compression performances. Specifically, using one set of the parameters in the simulation as an example:

(i)Take two distinct primes of similar bit lengths, $p = 11, q = 13$. Thus, $n = pq = 143$ and $\phi(n) = \phi(p)\phi(q) = n - (p + q - 1) = 120$.

(ii) Now we choose e, as part of the public key, such that $\gcd(e, \phi(n)) = 1$. We here choose $e = 7$. Thus, the public key, pbk=(n,e)=(143,7). Note that the choice of e is not unique and in practice, the smaller e is typically chosen, among all valid ones, the one in favor of computational simplicity.

(iii) Now we determine d, such that $1 \leq d < \phi(n)$, as the multiplicative inverse of e, with respect to $\phi(n)$, i.e. $de \equiv 1(\text{mod } \phi(n))$. It can be found that d=103. The private key is thus determined as prk=($\phi(n)$, d)=(120, 103).

Recall in Chapter 3, the encrypting algorithm of RSA first converts each plaintext symbol M into an integer m, such that $0 \leq m < n$ and $\gcd(m, n) = 1$. The necessity of such conversion comes from the fact that it's very likely that some of the plaintext symbols(i.e. M's) are not themselves co-prime with n. For each M, the choice of m is otherwise random(as long as all M's are converted to different m's) and there was no empirical indication in previous simulations that the choice of m for each character M matters significantly to the compression performances. Table 4.6 gives the ciphertext(in decimal and binary forms)for each integer M $\in \{0, \ldots, 15\}$.

We observe that the second and third bits(from the left), denoted as b_2 and b_3, respectively, satisfy:

$$\mathrm{P}_{00} = \frac{5}{16}, \mathrm{P}_{01} = \frac{3}{16} \tag{4.10}$$

$$\mathrm{P}_{10} = \frac{1}{4}, \mathrm{P}_{11} = \frac{1}{4}, \tag{4.11}$$

$$\Pr\{b_2 = 0\} = \Pr\{b_2 = 1\} = \frac{1}{2}, \tag{4.12}$$

$$\Pr\{b_3 = 0\} = \frac{9}{16} = 1 - \Pr\{b_3 = 1\}, \tag{4.13}$$

Where P_{ij} denotes $\Pr\{b_2 = i, b_3 = j\}$, for $i, j \in \{0, 1\}$.

character M	m	ciphertext=$m^e \pmod{n}$	binary representation of ciphertext
0	1	1	0 0 0 0 0 0 1
1	3	42	0 1 0 1 0 1 0
2	4	82	1 0 1 0 0 1 0
3	5	47	0 1 0 1 1 1 1
4	6	85	1 0 1 0 1 0 1
5	7	6	0 0 0 0 1 1 0
6	8	57	0 1 1 1 0 0 1
7	9	48	0 1 1 0 0 0 0
8	10	10	0 0 0 1 0 1 0
9	12	12	0 0 0 1 1 0 0
10	14	53	0 1 1 0 1 0 1
11	15	115	1 1 1 0 0 1 1
12	16	3	0 0 0 0 0 1 1
13	17	30	0 0 1 1 1 1 0
14	19	46	0 1 0 1 1 1 0
15	21	109	1 1 0 1 1 0 1

Table 4.6: (4,7) RSA Encryption

Thus, $\mathrm{P}_{00} \neq \Pr\{b_2 = 0\}\Pr\{b_3 = 0\}$, from which we conclude that b_2 and b_3 are no longer independent. Similar pairwise dependent relations can be found over many pairs within the seven bits in the ciphertext.

Moreover, let's see which encryption scheme results in 7-bit binary representations of the symbols' ciphertexts that are closer(in distribution) to independent sequences. Assuming symbols in the alphabet have uniform priors, i.e. $\pi(0) = \pi(1) = \ldots = \pi(15) = 1/16$. Let p_{ME} and p_R denote the joint distribution of the output 7-bit sequences of McEliece and RSA, respectively. Furthermore, use q_{ME}, q_R, respectively to denote the joint distribution of 7 independent bits, each has marginals according to Table 4.5 and 4.6. We compare KL divergences $D_{KL}(p_{ME}||q_{ME})$ as well as $D_{KL}(p_R||q_R)$ to measure which encryption scheme will result in an output sequence whose joint distribution is closer to a 7-bit independent binary sequence. Recall that, there would be no architectural loss, if the output sequence had joint density q_{ME} or q_R, repsectively.

There are only 16(out of a total of $2^7 = 128$) codewords $x \in \{0,1\}^7$ with $p_{ME}(x) = 1/16 \neq 0$. Moreover, since $q_{ME}(x) = 1/2^7$, for all $x \in \{0,1\}^7$, and each bit in the ciphertext of McEliece cryptosystem has equal marginals over 0 and 1. Thus,

$$D_{KL}(p_{ME}||q_{ME}) = 16 \cdot \frac{1}{16}\log_2(1/16)/(1/2^7) = 3$$

For the case of RSA, for the codewords y's with nonzero $p_R(y)$, we still have $p_R(y) = 1/16$, but for those y's, $q_R(y) \neq 1/2^7$. Instead, from Table 4.6, the marginals of each bit and $q_R(y)$ for each y can thus be calculated, with the assumption of those 7 bits are independent. After some tedious calculations, we find,

$$D_{KL}(p_R||q_R) = 3.432 > 3 = D_{KL}(p_{ME}||q_{ME})$$

Moreover, ten different sets of parameters(including the set above) of RSA(all mapping 4-bit plaintexts to 7-bit ciphertexts) give an average $D_{KL}(p_R||q_R) = 3.64$, which indicates that the binary representation of the ciphertext of McEliece cryptosystem is "closer" to an independent sequence than that of RSA. Therefore, the loss-free assumption is a worse one for RSA than for the McEliece cryptosystem and a larger performance loss should be expected for RSA. The above arguments offer one explanation on why the post-encryption compression performances are worse for RSA than McEliece cryptosystem. Later in Chapter 5, we will offer another perspective on the same issue and will subsequently argue these two perspectives are essentially consistent with each other.

On Symbolwise Encryption vs Blockwise Encryption

In all simulations of compressing encrypted large-alpahabet sources above, we encrypted the source symbolwisely – namely, each symbol in the large alphabet source is encrypted individually by the same encryption scheme with the same key. This makes the cipher easy to break. We could instead attempt to bind a number of symbols, say l, together as a block, and treat symbols in each block as a larger integer. Then, the new source will be the same kind, but with a larger alphabet. For example, if the source is generated as a 4-bit Markov source of order 1, but we choose to bind $l = 2$ symbols together as an 8-bit integer. Then the source becomes an 8-bit Markov source of order 1, with the transitional densities determined by those of the 4-bit sources. Therefore, the efforts of binding two 4-bit symbols and then encrypting the 8-bit symbols, have been incorporated in the attempts of compressing 8-bit Markov sources, where the best compression rates achievable, under different source entropies, have been provided. As can be seen from the previous simulations, there is slight performance improvements when compressing encrypted 8-bit sources, over compressing encrypted 4-bit sources, therefore, binding the symbols could potentially improve post-encryption compression performances. However, the complexity of the algorithm when encrypting over a larger alphabet is much higher than that when encrypting over a smaller alphabet. This can be seen from the fact that to implement the algorithm, there must be a translation table defined, which maps each symbol in the alphabet to its ciphertext(in binary form), based upon which message translations between layers are defined. The size of the translation table, which is bottlenecks the simplicty of the algorithm, increases geometrically with block length l. Therefore, if we were to bind a large number of symbols together for encryption, the (slight) advantage of post-encryption compression performances is significantly outweighed by the disadvantage of exploding complexity of the algorithm. Therefore, we have adopted the symbolwise encryption scheme.

Moreover, since the proposed algorithm essentially relies on messages passed iteratively between layers. Similar to message passing algorithms, good decoding performances will benefit greatly from the absence of short loops in the code subgraph, which is determined by the encoding matrix H. We will use the next subsection to describe the general methodology in finding such good encoding matrix H – the Progressive Edge Growth Algorithms.

4.3.2 On finding the (sub-)optimal encoding matrix H

Given the nominal compression rate and the (bitwise) length of the source, there are many possible constructions of parity check matrix H that can be used as encoding matrices. In general, to achieve the best compression performance, the **Progressive Edge Growth(PEG)**-construction [27][28] should be used. In this section, we will provide a brief introduction of the PEG algorithm.

Progressive Edge Growth Construction

In graph theory, *girth* refers to the length of the shortest cycle in a graph. For each symbol node(a.k.a source node) in the graph, a *local girth* can be defined as the length of the shortest cycle passing through that symbol node. Apparently, the girth of the graph is the smallest among all local girths. Since belief propagation algorithms generally work better if short cycles can be avoided, therefore constructing a Tanner graph with large girth is more than desirable for our application. Though constructing a Tanner graph of the given size with the largest possible girth is a rather difficult combinatorial problem, a sub-optimum algorithm to construct a Tanner graph with a relatively large girth is simple and feasible. PEG algorithm, where edges are constructed one by one such that local girth of each symbol node is maximized. Moreover, given the degree of each source node and the size of the matrix as inputs of the algorithm, the placement of each new edge is expected to have as small impact on the (graph) girth as possible. The fundamental idea is, for each source node, we attempt to find the most distant check node and then to place a new edge between the source node and this most distant check node.

Specifically, we construct edges for each source node sequentially. For source node s_j, when searching for the most distant check node to s_j, we need to generate the current subgraph from s_j. After we travel the subgraph down for depth l, denote $N_{s_j}^l$ as the set of check nodes included. Finally, two situations can occur:

(1). $N_{s_j}^l$ stops increasing and there are still some check nodes not included.

(2). $N_{s_j}^l$ includes all check nodes but $N_{s_j}^{l-1}$ does not.

Situation (1) usually occurs when the algorithm is in its early phase. The algorithm will choose one of the check nodes that are not currently reachable from s_j and create an edge between them. Under situation (2), all check nodes are reachable from s_j and the algorithm essentially chooses a check node from those that are not included in $N_{s_j}^{l-1}$ but are included in $N_{s_j}^l$, since these check nodes are the ones at the largest distance from s_j. When there are multiple check nodes in the above-stated set of candidates check nodes, we choose among them the one that has the smallest number of edges in the current graph setting. Furthermore, if there are more than one candidate check nodes that share the smallest number of edges, we can randomly choose one of them. The detailed algorithm can be found in **[27]** **[28]**and many other textbooks and is omitted here. In general, PEG algorithm generates a matrix with check node degree as regular as possible, which avoids overly large degrees on some of the check nodes, since strongly uneven degree distributions over check nodes have been shown empirically to worsen the performance of BP.

To generate an $m \times n$ binary PEG matrix H, in the worst case, the computational complexity and storage requirements scale as $O(mn)$ and $O(n)$, respectively. It outperforms Gallager's explicit construction for parity-check matrices corresponding to large girth graphs, which both of the computational complexity and the storage requirements, scale, in the best case, with $O(n^2)$.

The following theorem provides a lower bound of the girth of the graph corresponds to the PEG-generated low-density parity-check $k \times n$ matrix H:

Let (V, E) be an (in general) irregular Tanner graph in which d_c^{max} and d_s^{max}, respectively, are the largest degrees of the degree sequences D_c, and of the sequence of check node degrees and D_s, the sequence of source node degrees. Then the girth g of this graph is lower-bounded by

$$g \geq 2(\lfloor t_{low} \rfloor + 2), \tag{4.14}$$

where t_{low} is given by

$$t_{low} = \frac{\log(kd_c^{max} - \frac{kd_c^{max}}{d_s^{max}} - k + 1)}{\log[(d_s^{max} - 1)(d_c^{max} - 1)]} - 1 \tag{4.15}$$

which can be proved to outperform Gallager's construction with respect to large girth in the generated graph.

To give an idea of what the lower bound looks like, assuming we stick to the case of: $n = 2k$, with $d_{s_i} = 3$ for all source nodes s_i, $d_{c_a} = 6$ for all check nodes x_a, which constitutes a regular $(d_s, d_c) = (3, 6)$ PEG matrix. We wish to see how the girth changes with the number of check nodes k:

since $t_{low} = \log(6k - 2k - k + 1)/\log[(3-1)(6-1)] = \log(3k+1)/\log 10 = \log(3k+1)$, where the log is of base 10. Therefore,

$$g \geq 2(\lfloor t_{low} \rfloor + 2) = 2(\lfloor \log(3k+1) \rfloor + 2), \tag{4.16}$$

In our applications of interest, k has the order of 10^2 or 10^3, and we find from Figure 4-10 that PEG provides Tanner graphs of girth at least 8, which is in general satisfactory for our purposes of applications.

With the size of the matrix given, the tuning parameters for the PEG algorithm are the degree distributions of the source symbols. Given the average degree of the all source symbols, there are many possibilities of source degree distributions, each of which corresponds to a different version of PEG matrix and may result in different girth in the graph and finally different compression performances. There is no good theoretical results on the properties of the best source degree distributions, though PEG is known to be able to provide H good enough for our purposes of simulations.

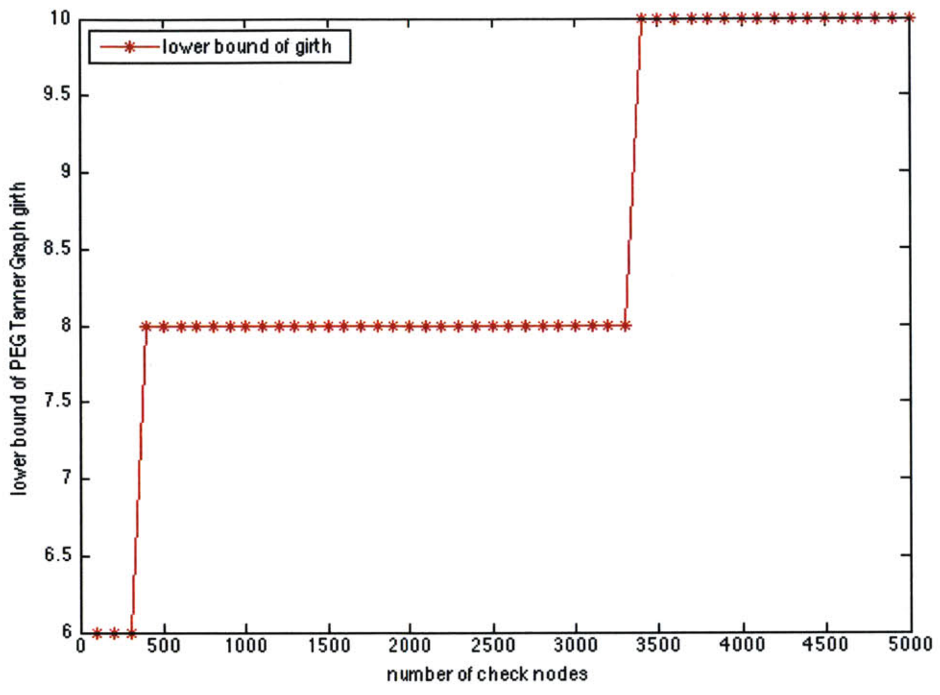

Figure 4-11: Girth of PEG Tanner Graph

On the gap between compression performances and source entropy rates

We note that, for all sources(see Figures 4-2 to 4-10), even including binary sources, where there is no architectural loss inherent in the algorithm, there is still a visible gap between the best compression rates achievable and the source entropy rates. According to **[7]**,.the gap does not result from any architectural loss but instead primarily from the non-optimal code selection. The loss could be reduced, however, only empirically. Assuming idealized, perfect code is used, the best compression rate achievable is referred to as the *BP decoding threshold*, denoted as ϵ^{BP}, in the sense that should the compression rate be below ϵ^{BP}, the BP algorithm will be sure to fail to converge to the correct result. There are literatures **[29]** that discover results of some good code selections to enable the best compression rate to approach ϵ^{BP}. However, since the major objective of this thesis was on the feasibility of compressing the encrypted data and the properties of encryption schemes, or more generally, properties of data-processing techiniques, that could potentially enable such good post-encryption(post-processing) compression performances, we are generally satisfied with any of the suboptimal H's generated by the PEG algorithms.

4.3.3 On optimal choice of doping

As briefly described before, some bits have to be doped so that the decoder can have some deterministic beliefs, on which the BP algorithm essentially relies for successful decoding. Without these deterministic beliefs, BP algorithms will likely lack the dynamics to converge to the correct solutions. Since doping will release the information, we are of course only interested in cases where only a small proportion(say, no more than 10%) of the bits need to be doped.

Given the number of bits to be doped as well as the encoding matrix H, the choice of which bits to dope can be optimized empirically in order to achieve the best possible compression performances. Some level of such efforts has been incorporated in all simulations in this chapter. In general, a closed-form, generalized optimal strategy on choosing the doping bits is intractable but there are some general guidelines one can follow in finding the best choices of doping:

(1) For compressing encrypted large-alphabet sources, it is better to dope all bits of a subset of symbols(or equivalently, to "dope some symbols") than to randomly dope the same number of bits. The reason behind it is simple – the fact that a subset of symbols are doped helps with some **reliable symbol-level macro-structures** being repetitively injected into the inference by the iterative nature of the decoding, which we believe has contributed to reducing the loss incurred at the message-translation layer.

(2) For compressing encrypted i.i.d. Bernoulli sources as well as Markov sources of order 1, large number of simulations indicate the manner of doping does not play a

visible role in the compression performances, so purely random doping is satisfactory. For Ising models, we believe it is desirable to avoid any un-doped bit to be adjacent with two or more doped bits. We are not able to analytically prove this observation but have firms believe this is true.

4.3.4 On Compression based on non-binary LDPC codes

During the course of this project, we have attempted to adapt the architecture into one based on non-binary LDPC codes so that no message translation is needed and the performance loss associated with the translation layer may thus be avoided. However, [**30**] states that the classical Belief Propagation algorithm over GF(q) has a computational complexity of $O(\mathrm{q}^2)$ and can be transferred into frequency domain, which scales down the complexity to $O(\mathrm{q}\log_2\mathrm{q})$. Therefore, in our application, if non-binary LDPC based compression is used, computational feasibility requires ciphertexts being kept extremely small, (since q has to be small), which is impossible for most real-life encryption schemes. Therefore, in this thesis, we have concentrated our attentions only to the architecture based on binary LDPC codes.

Chapter 5

Further Discussions

In Chapter 4, we showed simulation results of compressing encrypted data. In this chapter, we will further our discussion in Chapter 4 by noticing that encryption schemes can be seen as a special class of data processing techniques, where the plain-texts are processed by the encryption block in the system. In fact, in some other applications, source data needs to be processed in some way before it is compressed and transmitted and given a specific data processing operation, it is important to see whether a good compression performance is possible after such an operation. In this Chapter, we will focus on a subset of operations– linear operations and investigate what properties that a (linear) operation should have in order to enable good post-processing compression performances and argue that these properties are essentially in consistent with the analysis in Chapter 4.

5.1 Bit-level Sparsity of Linear Operations

In Chapter 4, an encryption operation can be seen as a block of source symbols, whose serialized binary representations are concatenated to form up bit sequence b of length B, are mapped into a bit sequence n, of bit length N, by some operation T : $\{0,1\}^B \rightarrow \{0,1\}^N$. **If T is constrained to be seen as a linear mapping**, then depending on the specific functionality of the encryption scheme, this mapping, as will be shown later, may depend on the input, which we will denote as T_b and refer to as an *inhomogeneous* linear operation. If T is not a function of input b, then we call such linear mapping *homogeneous.* Note that, in this chapter, homogeneousness is with respect to bit-level operations and should not be confused with that with respect to integer operations. In fact, all integer-based encrytion schemes must be homogeneous with respect to interger operations to be valid, but some of them may not be homogeneous in terms of bit-level operations. Moreover, for simplicity, in all descriptions to follow, **"operations(resp. mappings)" mean "linear operations(resp. mappings)"**.

For a general mapping $T : \{0,1\}^B \rightarrow \{0,1\}^N$, with abuse of notation, we borrow the concepts in encrytion schemes and refer to $\{0,1\}^B$ as the plaintext and $\{0,1\}^N$ as the ciphertext. In additional, we define $\lambda = (\lambda_1, \ldots, \lambda_B)$ as the degree distribution of each bit in the plaintext. Namely, λ_i is the number of bits in the ciphertext that b_i is incident to in the graph associated with mapping T. Finally, define the bit-level density of operation T, den(T), as den(T)=$\frac{\sum_i \lambda_i}{B}$.

Apparently, McEliece cryptostems, defined based on binary linear error-correcting codes, is homogeneous — given the public key $\hat{G}$, all possible plaintexts are mapped to the ciphertext, in the same way. This is also the case for all code-based cryptosystems. However, given the public keys, pbk=(n,e), RSA is inhomogeneous. We emphasize that any quantitative analysis of the compression performances of data either encrypted by RSA, or processed by any mapping that is not bitwise homogeneous from the persepective of their bit-wise sparsity, might not be completely scientifically sound, but some qualitative conclusions can be drawn nonethelessly.

Analysis on effects of Operation Density on Post-Operation Compression Performances for Homogeneous Operations

First, for data processing operations that are homogeneous, it is worth seeing what role the density of an operation plays in post-processing compression performances, since intuitively, given the encoding matrix H fixed, the more dense the pre-compression operation is, the more short loops may be created in the combined graph, which worsens the performances of loopy belief propagations. Ideally, joint optimization could be done over T and H, such that the number of short loops are minimized. However, in practice, we usually don't enjoy such degree of freedom– either T or H is fixed, or at least confined within a small subset, (for example, H is constructed by some sub-optimal algorithm, such as Progressive Edge Growth algorithm). Therefore, we aim at achieving some sub-optimal joint design– given LDPC parity-check matrix H, which is sub-optimal in its own right, we attempt to understand the role that den(T) makes in the post-processing compression performances and then identify the properties that operations should have for good post-processing compression performances.

We design the following simulations:

Simulations:

1. Definitions of Two Groups of Operations.

We define two groups, A and B, both of which include matrices that resemble encryption blocks. In Group A, we define $G_A^{(m)}$ $(3 \leq m \leq 6)$as the code generating matrix for $(2^m - 1, 2^m - m - 1)$ Hamming code, which has the size of $(2^m - m - 1) \times (2^m - 1)$. Note that, $G_A^{(m)}$ will have $2^m - m - 1$ columns of weight 1 and m columns of weight $2^{m-1} - 1$. We denote $G_A^{(m)} = [\mathrm{I}\ \mathrm{P}]$; where I is the $(2^m - m - 1) \times (2^m - m - 1)$ identity matrix and P is $(2^m - m - 1) \times m$, where each column has weight $2^{m-1} - 1$. Apparently, the mappings defined in Group A are all one-to-one.

With each $G_A^{(m)}$ defined, define its counterpart in Group B, $G_B^{(m)}$ (for $3 \leq m \leq 6$) as: $G_B^{(m)} = [\mathrm{H}_m\ \mathrm{P}^*]$, where H_m is of the size $(2^m - m - 1) \times (2^m - m - 1)$, and row i of $G_B^{(m)}$ is obtained by the modulo 2 sum of row i to row $2^m - m - 1$ of $G_A^{(m)}$. Therefore, we see that H_m is upper diagonal and thus rows of $G_B^{(m)}$ remain independent. Moreover, $\mathrm{Row}(G_B^{(m)}) = \mathrm{Row}(G_A^{(m)})$, where we use $\mathrm{Row}(\triangle)$ to denote the row space of the operation defined by matrix $\triangle$, since row space is preserved under elementary row operations. Therefore, the set of ciphertexts remains the same and thus the mapping defined by $G_B^{(m)}$ is also one-to-one.

To demonstrate, let's see the toy examples, $G_A^{(3)}$ and $G_B^{(3)}$:

$$G_A^{(3)}=\begin{pmatrix}1&0&0&0&1&1&0\\0&1&0&0&1&0&1\\0&0&1&0&0&1&1\\0&0&0&1&1&1&1\end{pmatrix},\ G_B^{(3)}=\begin{pmatrix}1&1&1&1&1&1&1\\0&1&1&1&0&0&1\\0&0&1&1&1&0&0\\0&0&0&1&1&1&1\end{pmatrix},$$

To calculate the density of operation $\text{den}(G_B^{(m)})$, we need to calculate the number of ones in P*. Since P is fixed for a given m, so is P*. There might not be generic results on the number of 1's in P*, as a function of m, but empirical results show that there is, at the very most, a rather mild decrease on the number of 1's in P*, compared to that in P. At the same time, as m gets larger, even just from $m = 3$ to $m = 6$, we can see the proportion of the number of columns in P(and in P*) takes in the total number of columns, $m/(2^m - 1)$, is rapidly vanishing and the impact of the potential disparity between the number of 1's in P* and that in P, has a rather small impact in the density of the operation. Therefore, we would hereafter make the assumption that the number of 1's in P* equals that in P for simplicity.

Therefore, the number of 1's in $G_B^{(m)}$, equals, $\frac{(1+2^m-m-1)\cdot(2^m-m-1)}{2} + m \cdot (2^{m-1} - 1) = \frac{(2^m-m)(2^m-m-1)}{2} + m \cdot (2^{m-1} - 1)$. Therefore, By definition,

$$\text{den}(G_A^{(m)}) = \frac{(2^m - m - 1)\cdot 1 + m\cdot(2^{m-1}-1)}{2^m - m - 1} = 1 + \frac{m\cdot(2^{m-1}-1)}{2^m-m-1}, \quad (5.1)$$

$$\text{den}(G_B^{(m)}) = \frac{\frac{(2^m-m)(2^m-m-1)}{2} + m\cdot(2^{m-1}-1)}{2^m-m-1} = \frac{2^m-m}{2} + \frac{m\cdot(2^{m-1}-1)}{2^m-m-1}, \quad (5.2)$$

As m gets large, we plot $\text{den}(G_A^{(m)})$ and $\text{den}(G_B^{(m)})$, versus m. Note that in Figure 5-1, only the points at integers m carry physical significance. However, we present continuous plots simply for a better demonstration of increasing rates of the density of operations with the increase of m.

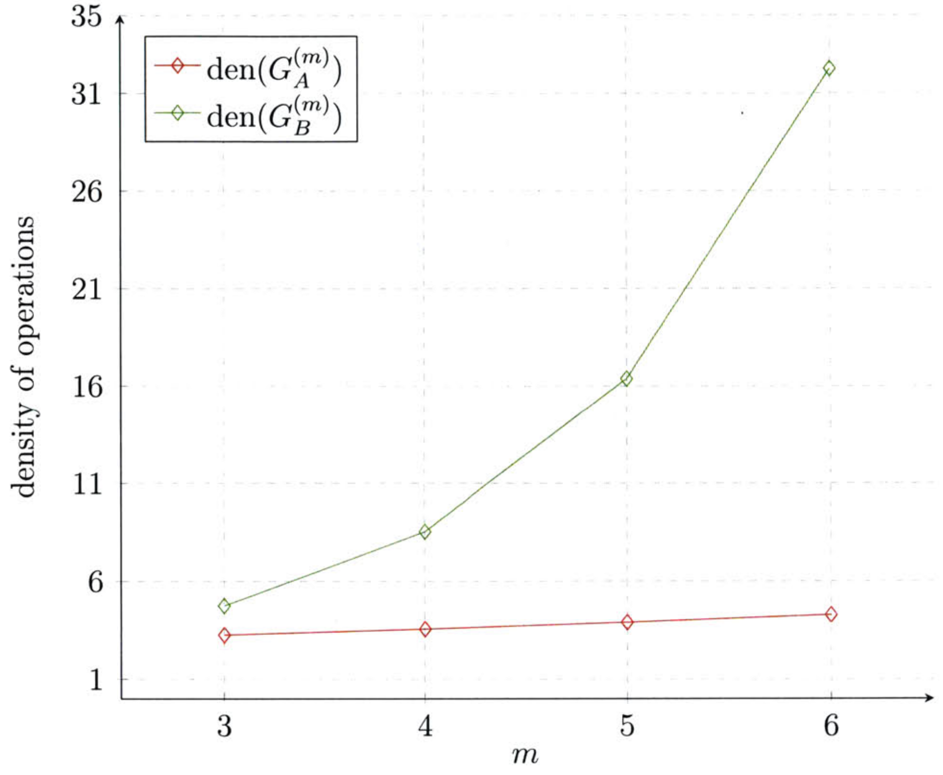

Figure 5-1: density of Group A and B matrices versus m

We can see the gap between den($G_B^{(m)}$) and den($G_A^{(m)}$), den($G_B^{(m)}$) $-$ den($G_A^{(m)}$) $= \frac{2^m-m-2}{2}$, which roughly doubles for each increment of m.

2. Post-operation Compression Performances vs Density of Operations.

In part 1, we showed that for each m, den($G_A^{(m)}$) and den($G_B^{(m)}$) will genenate the same set of outputs(or ciphertexts), with den($G_A^{(m)}$) being a more sparse operation while den($G_B^{(m)}$) being a more dense operation. We now compare post-operation compression performances of the two groups of operations to see how post-operation compression performances may vary with the density of processing operations.

Let the source $s = \{s_1, \ldots, s_n\}$ be 4-bit Markov source of order 1 with source length n=1000. Source alphabet is thus $\mathcal{A} = \{0, 1, \dots, 15\}$. Doping rate, $r_{dope} = 0.1$.

For a given m, we use $l(m)$ to denote the *processing block length*, i.e., the maximum number of source symbols whose concatenated binary reprsentations are jointly processed by $\text{den}(G_A^{(m)})$ (or $\text{den}(G_B^{(m)})$). Therefore, $l(m) = \lfloor (2^m - m - 1)/4 \rfloor$ and the remaining $(2^m - m - 1) - 4 \cdot l(m)$ positions in each block are filled up with some random bits, whose impact, already diminishing with the increase of m, can be completely eliminated by doping those bits in the compression process. For example, if $m = 4$, then each processing block consists of the concatenated binary representation of $l(4) = \lfloor (2^4 - 4 - 1)/4 \rfloor = 2$ symbols, followed by three random bits. Each block is then processed by $\text{den}(G_A^{(m)})$ and $\text{den}(G_B^{(m)})$, respectively, and the processed sequences are then compressed by progressive edge growth generated matrix H of the correct sizes. As usual, the best compression performances achievable, corresponding to $\text{den}(G_A^{(m)})$ and $\text{den}(G_B^{(m)})$ are provided in Figure 5-2 and Figure 5-3, respectively.

Observations and Comments:

1. All operations have negative effects of different extents on compression performances, as can be seen from the comparison between no-processing compression performances and the compression performances after any operations stated above.

2. We see that within each group (Group A or Group B), compression performance gets worse as m increases. There is only very mild loss in performance within group $G_A^{(m)}$, as m increases. However, there is much more visible compression loss within group $G_B^{(m)}$, as m increases. This is consistent with the fact that $\text{den}(G_A^{(m)})$ increases with m mildly while $\text{den}(G_B^{(m)})$ increases with m more drastically.

3. We notice that, the post-operation compression performances satisfy:

$$\text{per}(G_A^{(3)}) > \text{per}(G_A^{(4)}) > \text{per}(G_A^{(5)}) > \text{per}(G_A^{(6)}) > \text{per}(G_B^{(3)}), \tag{5.3}$$

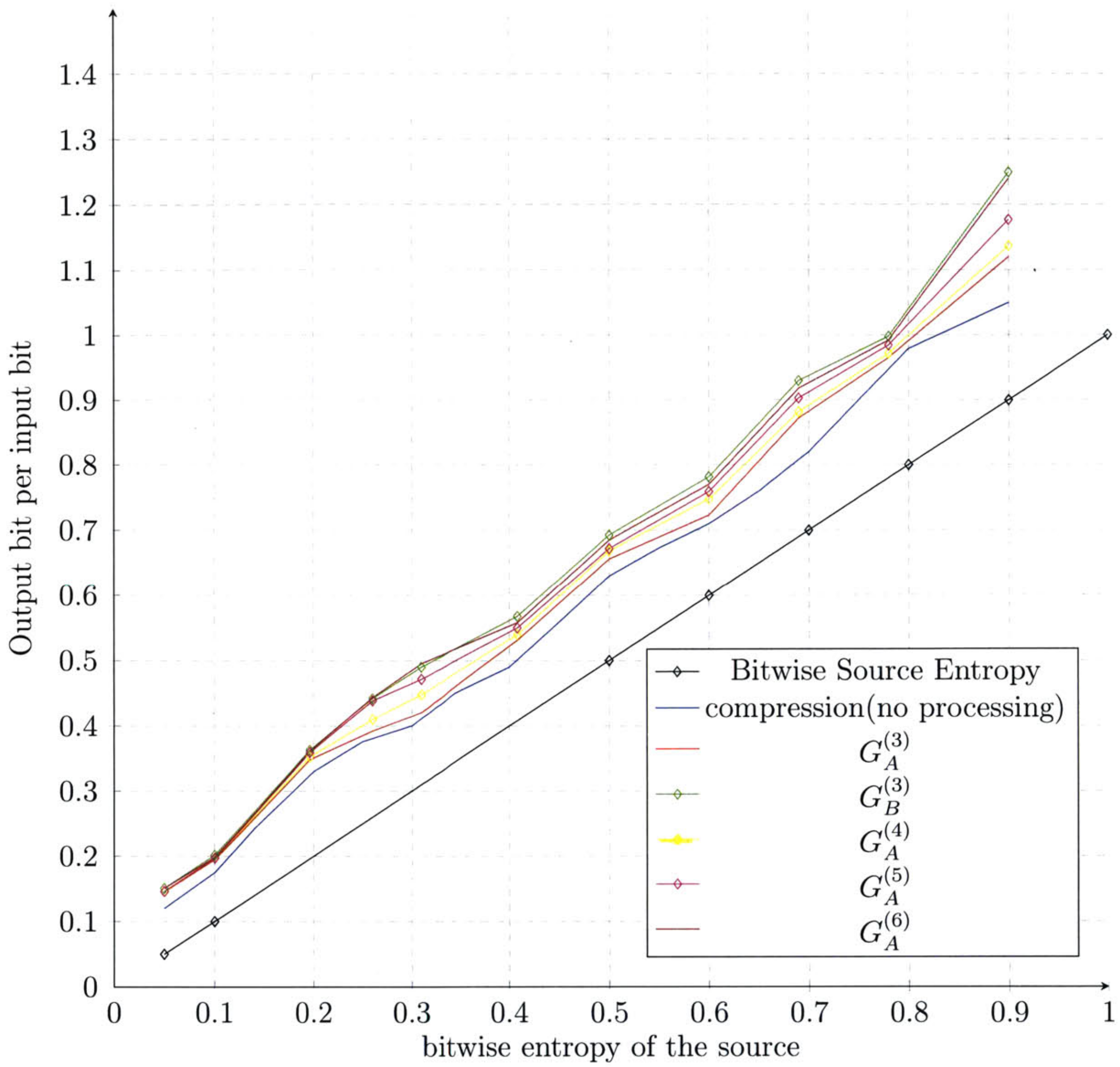

Figure 5-2: Operation Density on Compression Performance– Markov sources($G_A^{(m)}$)

and

$$\text{per}(G_B^{(3)}) > \text{per}(G_B^{(4)}) > \text{per}(G_B^{(5)}) > \text{per}(G_B^{(6)}), \tag{5.4}$$

where $\text{per}(\triangle_1) > \text{per}(\triangle_2)$ refers to the best achievable compression rates of the source processed by operation $\triangle_1$ are (almost) always smaller(a.k.a. the compression performances are always better) than those when the same source is processed by $\triangle_2$. Note that, the last inequality in (5.3) is not strict– there are several sample points(under several source entropies) where $\text{per}(G_B^{(3)})$ enables better performances than $\text{per}(G_A^{(6)})$(see the green curve and the purple curve representing performances

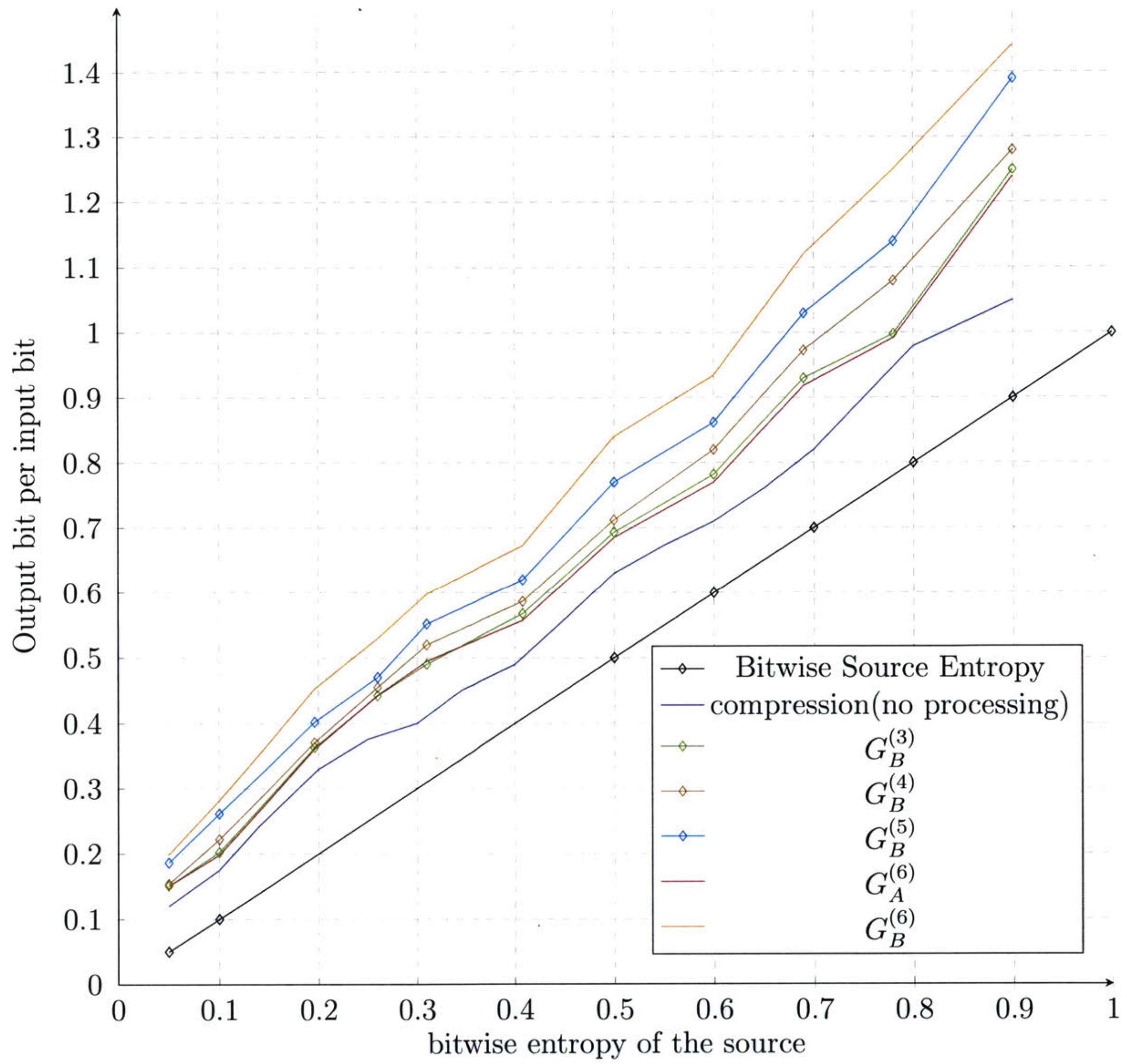

Figure 5-3: Operation Density on Compression Performance– Markov sources($G_B^{(m)}$)

of $G_B^{(3)}$ and $G_A^{(6)}$, respectively, in both Figure 5-2 and 5-3), though at more sample points, per($G_A^{(6)}$) enables better compression performances than per($G_B^{(3)}$) so that we still claim the last inequality as in (5.3). Note that, (5.3) is again consistent with the fact that, den($G_A^{(3)}$) < den($G_A^{(4)}$) < den($G_A^{(5)}$) < den($G_A^{(6)}$) < den($G_B^{(3)}$), with the gap for the last inequality being extremely small. We may conclude that for a homogeneous operation, its density is the most important factor in the post-operation compression performances. However, when two operations have densities that are sufficiently close to each other, other factors, for example, the compatibility of the specific form of an operation with the encoding matrix H, may play the most important role determining which operation will enable better post-operation compression performances.

5.2 Some further discussions on the performance of RSA

Recall simulation results in Chapter 4, where we see RSA-performances are inferior to McEliece-performances. However, since the bit-level (linear) operation underlying RSA is not homogeneous, it may not be scientifically strict to use the analysis in section 5.1. However, we may informally use the next toy example to demonstrate that, should we use the similar analysis of section 5.1 for RSA, we would find each output bit depends on each input bit within an encryption block, which is consistent with the inferior post-encryption compression performances of RSA.

Simulations:

We continue using the public key that we have used in Chapter 4, pbk=(n,e)=(143,7), for the RSA encrypting algorithm, and get the following plaintext-ciphertext mapping(Table 5-1):

Denote the plaintext as $p = [p_1\ p_2\ p_3\ p_4]$, and the ciphertext as $c = [c_1\ c_2\ c_3\ c_4\ c_5\ c_6\ c_7]$. Furthermore, use $p_{\sim i}$ to denote all other bits in p, except for p_i.

To determine whether p_i is dependent with c_j, we can do the following: fix a combination for $p_{\sim i}$, and see whether in any of those combinations, a flip of p_i will results in a flip of c_j. If for each possible $2^3 = 8$ combinations of $p_{\sim i}$, a flip of p_i does not result in a flip of c_j, then we conclude that p_i and c_j are independent. Otherwise, we

binary representation of character M	ciphertext	binary representation of ciphertext
0 0 0 0	1	0 0 0 0 0 0 1
0 0 0 1	42	0 1 0 1 0 1 0
0 0 1 0	82	1 0 1 0 0 1 0
0 0 1 1	47	0 1 0 1 1 1 1
0 1 0 0	85	1 0 1 0 1 0 1
0 1 0 1	6	0 0 0 0 1 1 0
0 1 1 0	57	0 1 1 1 0 0 1
0 1 1 1	48	0 1 1 0 0 0 0
1 0 0 0	10	0 0 0 1 0 1 0
1 0 0 1	12	0 0 0 1 1 0 0
1 0 1 0	53	0 1 1 0 1 0 1
1 0 1 1	115	1 1 1 0 0 1 1
1 1 0 0	3	0 0 0 0 0 1 1
1 1 0 1	30	0 0 1 1 1 1 0
1 1 1 0	46	0 1 0 1 1 1 0
1 1 1 1	109	1 1 0 1 1 0 1

Table 5.1: (4,7) RSA-Encryption Mappings

conclude they are dependent.

By the above test, we can conclude that (p_i, c_j) are dependent, for $\forall i \in \{1,2,3,4\}, \forall j \in \{1,\ldots 7\}$. Similar tests have also been conducted for a larger public key pbk=(n,e), where $n = pq = 61 \times 53 = 3233$ and $e = 17$. This will typically map a plaintext of length 6 to a ciphertext of length 12 and we can draw from the simulation results the same conclusion – within each encryption block of RSA, each bit of the ciphertext depends on each bit of the plaintext.

Brief Justification on the Unification of Arguments from Chapters 4 and 5

Finally, we argue that the argument we just provided is essentially consistent with the one in Chapter 4. Recall that in Chapter 4, we argued that the algorithm looks the problem of compressing encrypted data merely as an ordinary compression problem, with different symbol (and thus message) translations defined, depending on the functionality of the encryption scheme as well as the key used. Inherent in the algorithm, the independency among the bits in each symbol's binary representation is assumed for the algorithm to be completely lossless and we have seen that RSA typically generates a binary sequence that is far from(in the sense of distance of distributions) independent and thus more performance loss should be expected for RSA than for McEliece cryptosystem. This is in fact consistent with the argument we just presented in Chapter 5: if an operation is dense, like in RSA, where each output bit is dependent of each input bit within an encryption block, then it is readily imaginable that the output sequence of RSA is far from an independent sequence. Therefore, the arguments provided in Chapters 4 and 5 are in fact different perspectives of the same story.

Chapter 6

Conclusions and Future Work

In this thesis, we took advantage of a model-free compression structure to develop a practical algorithm to compress encrypted data. We argued that in theory, the algorithm should work as well as if no encryption is performed. Moreover, simulation results on encrypted synthetic and real world data were given. We also generalized encryption schemes to data-processing techniques and presented tests to illustrate the effects of sparsity of linear operations on the post-operation compression performances. Moreover, two arguments on the reason why compression performances are visibly different for RSA and McEliece system are provided in Chapters 4 and 5, respectively. We also argued that these two arguments are consistent with each other.

Note that, the model-free compression architecture, on which our work is based, is a lot more competitive when the source is in the low entropy regimes, which prevents the performances of compressing practically encrypted English files, which has medium-high entropy, from being extremely competitive. Moreover, text files need to be compressed losslessly, which may be overly demanding to the system, given the inevitable mismatch between real world text files and Markov models of English. However, many other signals in the real world are sparse in some domain and can be smartly compressed without degradation can be noticed, or even if the loss can be noticed, the loss does not outweigh the benefit of the significant reduction of size of signals. For example, such lossy compressions are more common in compressing multimedia signals, which may also need to be encrypted before they are compressed. Therefore, we will suspect the architecture be more competitive in compressing encrypted multimedia signals. This is one of the promising open problems to be addressed. Some efforts so far on model-free lossy compression can be found in **[7][32]**.

Some other open problems of this research include:

1. A deeper understanding of the reason why in all simulations above, the proposed algorithm is more competitive, i.e. the performance loss is smaller, in the low entropy region than in the high entropy region.
2. A deeper understanding of the effects of pre-compression operations in the compression performances. Most conclusions so far have been based on simulations results and are verified by designed tests. More analytical arguments are still needed.
3. Designs of encryption schemes, tailored to lossless and lossy model-free compression architectures based on LDPC codes are needed.

Bibliography

[1] M. Johnson, P. Ishwar, V.Prabhakaran, D. Schonberg, K. Ramchandran, "On Compressing Encrypted Data" *IEEE Transactions on Signal Processing*, vol. 52, No.10, Oct. 2004.

[2] D. Schonberg, S. Draper, K. Ramchandran, "On Blind Compression of Encrypted Correlated Data Approaching the Source Entropy Rate", *Proceedings of Allerton*, 2005.

[3] G. Blelloch,"Introduction to Data Compression", Jan 2013.

[4] C. Shannon,"A Mathematical Theory of Communication", *Bell System Technical Journal*, vol 27: 379-423, Oct 1948.

[5] T. Cover, J. Thomas, *Elements of Information Theory*, Wiley, 2nd Edition, ISBN:978-0-471-24195-9, July 2006.

[6] Y. Huang, G. Wornell, " A Class of Compression Systems with Model-free Encoding", *Information Theory and Applications Workshop*, Feb. 2014.

[7] Y. Huang, "Model-Code Seperation Architectures for Compression Based on Message-Passing", Ph.D. dissertation, Massachusetts Instititute of Technology, 2015.

[8] A. Ihler, J. Fisher, A. Willsky, "Loopy Belief Propagation: Convergence and Effects of Message Errors", *Journal of Machine Learning Research*, vol 6, 905-936, 2005.

[9] K. Murphy, Y. Weiss, M.Jordan,"Loopy Belief Propagation for Approximate Inference: An Empirical Study" *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*,1999.

[10] K. Murphy, *Machine Learning: a Probabilistic Perspective*, The MIT Press, Aug. 2012.

[11] M. Wainwright, M. Jordan,"Graphical Models,Exponential Families and Variational Inference", *Foundations Trends in Machine Learning*, vol.1, pp. 1-305, 2008.

[12] B. Cipra,"An Introduction to the Ising Model", Dec.1987.

[13] R. Gallager, "Low-density parity-check codes"*IRE Transaction on Information Theory*, vol.8, no.1, pp.21-28,Jan 1962.

[14] W. Ryan, S. Lin, *Channel Codes, classical and modern*, Cambridge University Press, Oct. 2009.

[15] R. Rivest, A. Shamir, L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems",*Communications of the ACM*, vol.21,no.2, pp.120-126, 1978.

[16] A.Kak, "Public-Key Cryptography and the RSA Algorithm" , *Lecture 12, Computer and Network Security*, April 22, 2015.

[17] R. McEliece, "A Public-Key Cryptosystem Based on Algebraic Coding Theory", 1978.

[18] S. Au, C. Eubanks-Turner, J. Everson, " The McEliece Cryptosystem", Sep. 2003.

[19] L. Onsager, "Crystal Statistics, I. A Two-Dimensional Model with An Order-Disorder Transition" , *Physics Review* vol. 65, 117149, 1944.

[20] D. Mackay, "Introduction to Monte Carlo Methods", 1996.

[21] N. Wagner, *The Laws of Cryptography with Java Code*, June. 2003.

[22] *Theory of Data Compression* www.data-compression.com/english.shtml.

[23] C.Darwin, "Origin of Species", Nov. 1859.

[24] C. Bronte, "Jane Eyre", Oct. 1847.

[25] S. Johnson, "Introducing Low-Density Parity-Check Codes", ACoRN Spring School, 2007.

[26] C. Shannon, " Prediction and Entropy of Printed English", *Bell Systems Techinical Journal*, vol. 30, pp. 50-64, Jan. 1951.

[27] X.Hu, E. Eleftheriou, D.Arnold, "Regular and Irregular Progressive Edge-Growth Tanner Graph" *IEEE Transactions on Information Theory*, VOL. 51, No.1, Jan. 2005.

[28] D. Mackay, *David Mackay's Gallager Code Resources*, Aug. 2008.

[29] T. Richardson, M. Shokrollahi, R. Urbanke, "Design of Capacity-approaching irregular low-density parity-check codes", *IEEE Transaction on Information Theory*, vol.47, no.2, pp. 619-637,2001.

[30] V. Ganepola, R. Carrasco, I. Wassell, S. Goff, "Performance Study of Non-binary LDPC Codes over GF(q)", *Proceddings of CSNDSP08*, pp.585-589, 2008.

[31] J. Martinez-Mateo, D. Elkouss, V. Martin, "Improved Construction of Irregular Progressive Edge-Growth Tanner Graph", *IEEE Communication Letters*, Vol.14, Issue 12, PP. 1155-1157, Oct 2010.

[32] Y.Huang and G.Wornell, "Seperation Architecture for Lossy Compression", *Proceedings of IEEE Information Theory Workshop*, April 2015.

[33] T. Berger, *Rate Distortion Theory,:Mathematical Basis for Data Compression*, Prentice Hall, Oct. 1971.

[34] D.Koller, N.Friedman,*Probabilistic Graphical Models–Principles and Techniques*, The MIT Press, July, 2009.

[35] B. Kaliski, " The Mathematics of the RSA Public-Key Cryptosystem", 2006.

[36] B.Potetz, T. Lee, "Efficient Belief Propagation for higher order cliques using linear constraint nodes",*Computer Science Department*, May 2008.

[37] X. Zhang, "A Sampling Technique Based on LDPC Codes", Master of Science Thesis, MIT, Feb 2015.

[38] H. Delfs, H. Knebl, *Introduction to Cryptography*, 2nd Edition, Springer, May 2007.

[39] F. Kschischang, B. Frey, H.Loeliger, "Factor Graphs and the Sum-Product Algorithm", *IEEE Transactions on Information Theory*,vol.47, no.2., Feb. 2001.