Master Thesis

University of Bergen

Department of Informatics

Tail-biting Codes for Lattice Wiretap Coding

Author:
Palma Rud Persson

Supervisors:
Maiara Francine Bollauf,
Hsuan-Yin Lin,
Øyvind Ytrehus

June 1, 2023

Acknowledgements

I would like to start by thanking my supervisors, Maiara Bollauf, Hsuan-Yin Lin, and Øyvind Ytrehus, for their support, guidance, and valuable insights throughout my thesis.

I would also like to thank my fellow students at the algorithm study hall and the bachelor study hall, for supporting me through ups and downs, and generally making this an enjoyable and entertaining year.

Finally, I would also like to thank my family and friends for being there for me. A special thanks to my sister, Amanda, who is a master at creating figures and my partner, Tines, who let me use his computer to run my code.

Palma Rud Persson
June 1, 2023

Abstract

The secrecy gain of Construction A lattices obtained by tail-biting rate $1/2$ convolutional codes is studied to evaluate the secrecy performance of a lattice in a wiretap channel communication. The higher the secrecy gain, the more difficult for an eavesdropper to correctly guess the transmitted message. To explore the best potential secrecy gain, an exhaustive search is conducted, considering various constraint lengths and even code lengths up to 100. The methods' validity is verified by evaluating the minimum free distance and comparing the results with known findings. This investigation identifies outcomes close to the upper bound of unimodular lattices.

List of Figures

List of Tables

Contents

Nomenclature

$\boldsymbol{x}$	Vectors
$\mathscr{C}$	Linear block code
$\mathcal{C}$	Convolutional code
n	Code length
d	Minimum distance
d_{free}	Minimum free distance
q	Field size, for binary $q = 2$
G	Generator matrix of a code
$\mathsf{G}(D)$	Generator matrix on canonical form
R_{b}	Code rate of a linear block code
R	Code rate of a convolutional code
h	Length of a input sequence
Λ	Lattice
B	Generator matrix of a lattice
$\mathbb{Z}^n$	Cubic lattice
$\Lambda_{\text{A}}(\mathscr{C})$	Construction A lattice obtained from a linear code $\mathscr{C}$
$\Theta_\Lambda(z)$	Theta series of a lattice
Ξ_Λ	Secrecy function of a lattice

Chapter 1

Introduction

1.1 Motivation

Physical layer security has become increasingly important in recent years due to the spread of wireless and mobile communication devices. As these devices become more prevalent in our daily lives, the need for secure communication channels becomes more critical. With the rollout of newer technologies, such as 5G technology, new challenges and opportunities have presented themselves. With 5G, communication networks are becoming more complex and diverse, which means that traditional security methods may not be sufficient.

One can approach this issue in two ways: through computational security or information-theoretic security [3]. Computational security is based on the assumption that certain mathematical problems are computationally hard to solve. This approach aims to ensure that an attacker cannot break the encryption by computing the key or message in a reasonable amount of time [4]. One example that may achieve this is lattice-based cryptography, which relies on the computational hardness of certain problems in lattice theory [5, 6].

Information-theoretic security is also based on mathematical principles; however, this approach aims to guarantee perfect secrecy by ensuring that the intercepted message does not contain any information about the original message [3]. Lattice-based coset coding is one such technique that can provide secure and reliable communication, as described in [7] and [8]. It has been discovered that the theta series of a lattice can serve as a quality measure for good wiretap lattice codes in the context of the Gaussian wiretap channel. The secrecy function expresses this measure, and to achieve optimal security, the (strong) secrecy gain must be maximized in order to reduce the probability of successful decoding by eavesdroppers.

In a previous study by Belfiore and Solé [9], they discovered the symmetry point of the secrecy function for unimodular lattices. Building upon this, Bollauf, Lin, and Ytrehus put forward a conjecture in their recent work [10, 11] that the secrecy gain for *formally unimodular lattices* can be achieved at this symmetry point. They also proposed a universal method for determining the strong secrecy gain of formally unimodular periodic packings.

This thesis extends the research by exhaustively searching different parameters to identify the block codes constructed by tail-biting convolutional codes that maximize the secrecy gain.

1.2 Objective

The primary goal of this thesis is to evaluate the secrecy function of block codes obtained by tail-biting convolutional codes while also checking our methods using the free distance metric. This is a well-known and well-researched metric and is commonly used to, among other things, detect errors in codes. To evaluate the secrecy function, we aim to find the global maximum, also called the secrecy gain. We will employ an exhaustive search of various tail-biting convolutional codes with different parameters to identify the code with the highest possible secrecy gain for those parameters. In addition, we will validate our methods by utilizing known results of free distance. By comparing our results with those obtained using other methods, we can evaluate the accuracy and reliability of our findings.

1.3 Thesis Organization

The thesis is organized as follows. Chapter 2 gives an overview of relevant key concepts from coding theory, convolutional codes, and lattices. We continue with work related to this thesis in Chapter 3; we discuss previous work on free distance for convolutional codes and introduce key concepts of secrecy gain. Chapter 4 presents our results and discusses the methods used to find them, such as implementation and test data. Chapter 5 summarizes and concludes our findings and discusses possible future work.

1.4 Outcome

The results of this paper were submitted for a possible publication and presentation at the 2023 International Symposium of Topics in Coding (ISTC) [12].

Chapter 2

Background

2.1 Linear block codes

Coding theory is a branch of mathematics and computer science that deals with the efficient and reliable transmission of information over a communication channel. It has become an essential component of information security as it provides a set of tools to ensure the confidentiality, integrity, and availability of information.

One of the fundamental concepts of coding theory is error-correcting codes, which detect and correct errors during data transmission. These codes add redundancy to the transmitted data, allowing the receiver to detect errors and correct them without retransmitting the entire message [2]. Block codes are error-correcting codes that divide the message into fixed-size blocks and apply an encoding function to each block. The resulting encoded blocks are then transmitted over the communication channel. The definitions here are based on [2].

Definition 2.1.1 (Linear block code)**.** A q-ary $[n, k]$ linear code $\mathscr{C}$ of length n is a subspace over a finite field $\mathbb{F}_q$, so $\mathscr{C} \subseteq \mathbb{F}_q^n$, where q is the field size. E.g. for binary $q = 2$. The dimension of $\mathscr{C}$, k, is the number of linearly independent vectors in the subspace. The rate of a linear block code is $R_\mathrm{b} = k/n$.

Definition 2.1.2 (Hamming weight and Hamming distance [13])**.** Given a code $\mathscr{C}$ over a finite field $\mathbb{F}_q$, the Hamming weight of a codeword $\boldsymbol{c} \in \mathscr{C}$ is the number of non-zero symbols in the codeword $\boldsymbol{c} \in \mathscr{C}$, and the Hamming Distance between two codewords of length n, $\boldsymbol{u} = (u_1, \cdots, u_n)$ and $\boldsymbol{v} = (v_1, \cdots, v_n)$, denoted $d(\boldsymbol{u}, \boldsymbol{v})$, is the number of positions at which the two codewords differ, i.e.,

$$d(\boldsymbol{u}, \boldsymbol{v}) = |\{i \colon u_i \neq v_i, i = 1, 2, \ldots, n\}| .$$

Definition 2.1.3 (Minimum distance of a code $\mathscr{C}$)**.** The minimum distance is the minimum Hamming distance between any two distinct codewords of length n, $\boldsymbol{u} = (u_1, \cdots, u_n)$ and $\boldsymbol{v} = (v_1, \cdots, v_n)$ in a code $\mathscr{C}$, i.e.,

$$d(\mathscr{C}) = \min\{d(\boldsymbol{u}, \boldsymbol{v}) : \boldsymbol{u}, \boldsymbol{v} \in \mathscr{C}, \boldsymbol{u} \neq \boldsymbol{v}\}.$$

The minimum distance is an important parameter of a code as it determines the maximum number of errors it can detect and correct. Specifically, a code with

minimum distance d can detect up to $d-1$ errors while it can correct up to $\lfloor (d-1)/2 \rfloor$ errors. A code with a larger minimum distance can correct more errors and is, therefore, more reliable in noisy communication channels [2, p. 76].

Definition 2.1.4. An $[n, k]$ linear code $\mathscr{C}$ can also be denoted as an $[n, k, d]$ linear code $\mathscr{C}$, where d is the minimum Hamming distance between any two distinct codewords in $\mathscr{C}$.

Definition 2.1.5 (Weight enumerator of a code $\mathscr{C}$)**.** The weight enumerator of an $[n, k, d]$ code $\mathscr{C}$ describes the Hamming weight of all codewords in $\mathscr{C}$. Specifically, let $A_w(\mathscr{C})$ be the number of codewords of Hamming weight w in $\mathscr{C}$, where $w \in \{0, 1, \ldots, n\}$. Then, the weight enumerator of a code $\mathscr{C}$ is defined as

$$W_{\mathscr{C}}(x, y) = \sum_{w=0}^{n} A_w(\mathscr{C}) x^{n-w} y^w.$$

Definition 2.1.6 (Basis of a linear block code)**.** Let $\mathscr{C}$ be a linear block code of length n and dimension k over a finite field $\mathbb{F}_q$. A basis of $\mathscr{C}$ is a set of k linearly independent codewords $\boldsymbol{v_1}, \boldsymbol{v_2}, \cdots, \boldsymbol{v_k}$ that span the code, i.e., every codeword in $\mathscr{C}$ can be expressed as a linear combination of the basis vectors:

$$\forall\, \boldsymbol{c} \in \mathscr{C}, \exists\, a_1, a_2 \ldots, a_k \in \mathbb{F}_q \colon \boldsymbol{c} = a_1 \boldsymbol{v_1} + a_2 \boldsymbol{v_2} + \cdots a_k \boldsymbol{v_k}.$$

Definition 2.1.7 (Generator matrix of a linear block code)**.** A basis of $\mathscr{C}$ can be represented by a $k \times n$ generator matrix G whose rows are the basis vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$, i.e.,

$$\mathsf{G} = \begin{bmatrix} \boldsymbol{v_1} \\ \boldsymbol{v_2} \\ \vdots \\ \boldsymbol{v_k} \end{bmatrix}.$$

The generator matrix G is used to encode the k information bits into the n codeword bits, and it also allows us to express any codeword in $\mathscr{C}$ as a linear combination of the basis vectors:

$$\forall\, \boldsymbol{c} \in \mathscr{C}, \exists\, \boldsymbol{a} \in \mathbb{F}_q^{1\times k} \colon \boldsymbol{c} = \boldsymbol{a}\mathsf{G}.$$

Remark 1. The choice of basis is not unique, and many bases can generate the same linear code. The only criteria are that there are k independent vectors generating the basis and that the basis can be transformed into each other using a nonsingular matrix S, meaning the determinant $\det(\mathsf{S}) \neq 0$.

2.2 Wiretap Channel

In information theory, a channel is a communication path between a sender and a receiver through which information is transmitted. The channel may be physical (e.g., a wire, a fiber optic cable, or the airwaves) or abstract (e.g., a mathematical model of signal distortion). The properties of a channel, such as its capacity, noise level, and error rate, determine the maximum rate at which information can be reliably transmitted through it. Figure 2.1 below describes a typical system of information signals for use in a data transmission or storage system. A message is produced by the information source, which can be everything from a computer to a human speaker. It is then encoded to digital form in the source encoder, changing the message into a convenient form for the channel encoder. This may include data compression, such that all transmitted message bits are essentially independent and identically distributed. The channel encoder then proceeds to encode it into code signals, which makes the message ready to be sent over the channel. A channel can be everything from a data transmission channel to a storage device. Examples of data transmission channels are things such as an optical fiber, a microwave radio link, and a copper telephone wire, and examples of a data storage device can be a magnetic or optical disk. The channel may also introduce noise, which can affect the channel decoder and the source decoder when trying to decode the message for the destination [1, p. 64].

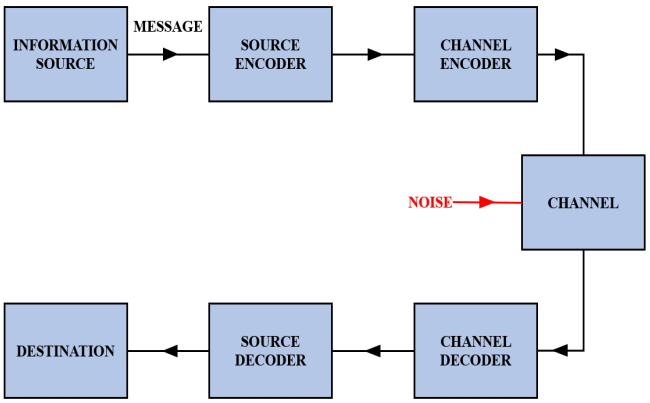

Figure 2.1: Diagram of a possible channel [1, p. 64].

A wiretap channel is a specific type of communication channel in which an eavesdropper, known as the wiretapper, may intercept some or all of the transmitted messages without the sender or the intended receiver being aware of it. The goal here is that the receiver needs the codewords to be decoded efficiently while still keeping the eavesdroppers oblivious about what is being communicated. Convolutional encoders are often used here because they have efficient decoding methods like the Viterbi decoding algorithm.

This work studies the Gaussian wiretap channel only.

2.3 Convolutional Codes

A convolutional code $\mathcal{C}$ is an error-correcting code introduced in the 1950s as an alternative to block codes [14]. The main difference between these codes is that the convolutional encoder contains memory and that, at any given time unit, the output depends on both the current and earlier inputs. Now let us get more into what a convolutional encoder is.

A convolutional encoder has a rate $R = k/n$, meaning that for every k bits in the input, the encoder will produce n outputs. E.g., if $R = 1/2$, the encoder will produce two output bits for every input bit. The outputs are defined in the generator matrix $\mathsf{G}_{\mathcal{C}}$, where m is the number of memory elements and $\mathsf{G}_0, \cdots, \mathsf{G}_m$ are generator submatrices [2, p. 453],

$$\mathsf{G}_{\mathcal{C}} = \begin{bmatrix} \mathsf{G}_0 & \mathsf{G}_1 & \cdots & \mathsf{G}_{m-1} & \mathsf{G}_m & & & & \\ & \mathsf{G}_0 & \mathsf{G}_1 & \cdots & \mathsf{G}_{m-1} & \mathsf{G}_m & & & \\ & & \mathsf{G}_0 & \mathsf{G}_1 & \cdots & \mathsf{G}_{m-1} & \mathsf{G}_m & & \\ & & & \ddots & \ddots & & & \ddots & \\ & & & & \mathsf{G}_0 & \mathsf{G}_1 & \cdots & & \mathsf{G}_m \\ & & & & & \mathsf{G}_0 & \mathsf{G}_1 & \cdots & \mathsf{G}_{m-1} \end{bmatrix}.$$

Example 1. Consider $R = 1/2$ convolutional code with the generator submatrices:

$$\mathsf{G}_0 = (101), \quad \mathsf{G}_1 = (111).$$

has the generator matrix $\mathsf{G}_{\mathcal{C}}$:

$$\mathsf{G}_{\mathcal{C}} = \begin{bmatrix} 101 & 111 & & \\ & 101 & 111 & \\ & & \ddots & \ddots \end{bmatrix}.$$

The generator matrix can also be written in polynomial form, with *generator polynomials* $g_0(D), g_1(D), \ldots, g_{m-1}(D)$, where each polynomial represents an output according to the state of the encoder as well as the input, i.e.,

$$\mathsf{G}(D) = [g_0(D), g_1(D), \cdots, g_{m-1}(D)].$$

Example 2. If we use the same code as in Example 1, where $\mathsf{G}_0 = (101)$ and $\mathsf{G}_1 = (111)$, that can be written as

$$g_0(D) = 1 + D^2, \quad g_1(D) = 1 + D + D^2.$$

So we have the generator written on polynomial form $\mathsf{G}(D)$:

$$\mathsf{G}(D) = [1 + D^2, 1 + D + D^2].$$

The constraint length of a convolutional code is defined as the sum of the lengths of all k shift registers, while the number of memory elements is the longest shift register of all k shift registers [2, p. 459]. In this thesis, we only look at $R = {}^{k}/_{n} = {}^{1}/_{2}$, so constraint length and the number of memory elements are the same and will be noted m. The shift register receives k input bits for each time t. So the total bits in the shift encoder, the memory elements, and the input bit will be $m + 1$. The figure underneath shows a rate $R = {}^{1}/_{2}$ convolutional encoder with $m = 3$ memory elements and generator $\mathsf{G} = [1011, 1111]$, where $\boldsymbol{u}$ is the input and $\boldsymbol{v_0}$ is the output for the first generator submatrix $\mathsf{G}_1 = 1011$ and $\boldsymbol{v_1}$ is the output for the second generator submatrix $\mathsf{G}_2 = 1111$.

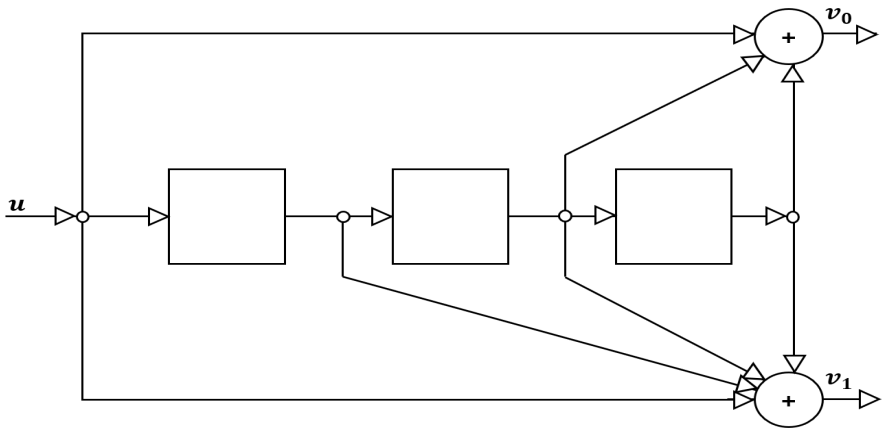

Figure 2.2: A rate $R = {}^{1}/_{2}$ feedforward convolutional encoder with $m = 3$ and generator $\mathsf{G} = [1011, 1111]$ [2, p. 455].

The convolutional encoder depends on time. Each time unit, the encoder takes the current state and the input bit and evaluates it according to the generator matrix, producing an output. The input bit at the time t will shift at time $t + 1$ and become a memory element.

The convolutional encoder has at any given time a *state*. This state describes the current memory elements. There are 2^m states. In our Example 2 above, the encoder can have the following $2^2 = 4$ different states

$$00, 01, 10, 11.$$

However, even though the encoder starts at a finite time, it can produce indefinitely long encoded sequences or codewords. If the encoder has no earlier memory i.e., at the start of the encoding process, the encoder usually begins in the all-zero state. In our example, that would be the state 00 [2, p. 470].

To facilitate the decoding, which we will discuss later, it is also necessary to return the encoder to the original state. This is done by entering the terminating bits, which are of the same length as the number of memory elements, into the encoder until the encoder returns to the original state. For feedforward convolutional encoders, the original state is usually the all-zero state, so terminating the encoder is done by entering m 0-inputs into the encoder. The terminating bits are fixed according to the original state, so they cannot be a part of the information sequence [2, p. 485].

By terminating the convolutional code, we can look at the code as an $[N, K^*]$ block code with a $K^* \times N$ generator matrix, where

$$K^* = kh, \quad N = n(h+m),$$

given that n is the number of outputs per input, h is the length of the input sequence, m is the number of terminating bits, and k is the number of inputs processed together.

For such an $[N, K^*]$ terminated convolutional code, the ratio of the number of corresponding information bits to the length of the corresponding codeword gives the *block code rate* obtained by terminating the convolutional code [2, p. 486], which is defined as

$$R_{\mathrm{b}} = \frac{K^*}{N} = \frac{kh}{n(h+m)}.$$

When the length of the input sequence is quite large compared to the length of the terminating bits, that is, if $h \gg m$, then

$$\frac{h}{(h+m)} \approx 1,$$

and the block code rate is approximately equal to the convolutional code rate, i.e., $R \approx R_{\mathrm{b}}$. However, when h is small, R_{b} is reduced by a fractional amount called the fractional rate loss R_{L}, which is

$$R_{\mathrm{L}} = \frac{\frac{k}{n} - \frac{kh}{n(h+m)}}{\frac{k}{n}} = \frac{m}{h+m}.$$

For longer sequences, this loss is quite small. E.g. for a code with $h = 1000$ and $m = 3$, $R_{\mathrm{L}} = {}^{2}/_{1002} \approx 0.02\%$. While for shorter sequences, with, for example, $h = 10$, $R_{\mathrm{L}} = {}^{3}/_{13} \approx 23.1\%$. One way to eliminate this loss is by not transmitting the terminating bits. However, this is shown to worsen the code's performance. Another way to eliminate this loss is to choose the initial state carefully and terminate the code by returning to the same origin state [2, p. 486]. This method is called tail-biting, and we'll discuss this further in Section 2.4.

2.3.1 Catastrophic encoders

A convolutional generator is considered catastrophic if an information sequence has infinitely many nonzero digits sent into an encoder and then decoded into a codeword with only finitely many nonzero digits [15, p. 56].

Example 3 ([15, p. 57])**.** Consider the generator $\mathsf{G} = [1 + D^3, 1 + D + D^2 + D^3]$ for a rate $R = {}^{1}/_{2}$ convolutional code. With the information sequence $u(D) = {}^{1}/_{(1+D)} = 1 + D + D^2 + D^3 \cdots$ with weight $w_u = \infty$. The codeword, however, $v(D) = u\mathsf{G} = [1 + D + D^2, 1 + D^2] = [1, 1] + [1, 0]D + [1, 1]D^2$ which has weight $w_v = 5 < \infty$.

While decoding, if only five well-placed errors occur, this might be decoded to the all-zero codewords. Resulting in the loss of infinitely many information bits.

A more formal definition of catastrophic encodes is given below. Note that only the generators can be deemed catastrophic, as convolutional codes can have many both catastrophic and non-catastrophic generators.

Proposition 1 ([2, p. 483])**.** Given a convolutional code $\mathscr{C}$ generated by $\mathsf{G}(D)$. $\mathsf{G}(D)$ has a feed-forward inverse $G^{-1}(D)$, if and and only if:

$$\gcd[g_0(D), g_1(D), \ldots, g_k(D)] = D^l,$$

for $l \geq 0$ and, where gcd denotes the *greatest common divisor*.

Proposition 2 ([2, p. 483])**.** $\mathsf{G}(D)$ is a catastrophic encoder if and only if it does not have a feed-forward inverse $\mathsf{G}(D)^{-1}$.

2.3.2 Trellis diagram and State tables

A state table for, a convolutional code, represents all possible states and which inputs lead to what output, as well as what the next state is, for each state. The state table has many possible variations, and they show more or less information depending on what is needed. Underneath is a description of a type of state table.

- m_0 describes the input bit.
- m_1 and m_2 are the current memory elements, and together they describe the current state
- o_1 and o_2 describe the output
- The next state is the input bit together with m_1

Using the example earlier, with a $1/2$ convolutional encoder where the generator matrix $\mathsf{G}_\mathcal{C}$ is:

$$\mathsf{G}_\mathcal{C} = \begin{bmatrix} 101 & 111 & & \\ & 101 & 111 & \\ & & \ddots & \ddots \end{bmatrix}.$$

We then obtain the state table:

input/m_0	m_1	m_2	o_1	o_2	next state
0	0	0	0	0	00
1	0	0	1	1	10
0	0	1	1	1	00
1	0	1	0	0	10
0	1	0	0	1	01
1	1	0	1	0	11
0	1	1	1	0	01
1	1	1	0	1	11

Table 2.1: State table for $\mathsf{G} = [101, 111]$

Another functional representation of the convolutional encoder is a Trellis diagram, first introduced by Forney in [16]. As seen in Figure 2.3 below. The nodes represent the states, with the arrows pointing at the next state given the input. Here the gray arrow represents the 0 inputs, and the black represents the 1 inputs. The outputs are either written along the line pointing to the next state or, as is done here, written to the nodes' left. It is written in the format input/output.

The trellis below is the same as demonstrated earlier with the state table. The states are written in decimal, so 0 corresponds to state 00, 1 to 01, and so on.

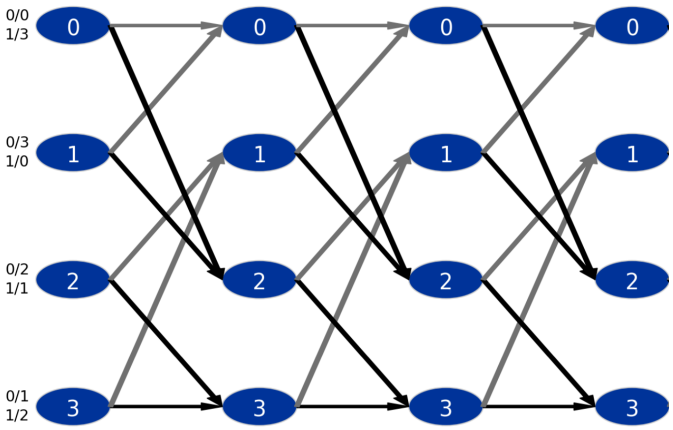

Figure 2.3: Trellis for $\mathsf{G} = [101, 111]$

2.3.3 Viterbi decoding

The Viterbi decoding algorithm was designed in 1967 by Viterbi [17] and is used for decoding convolutional codes. The algorithm works by finding the most likely sequence of encoded symbols that could have been transmitted, given the received sequence of symbols and the properties of the code. This is accomplished by computing a set of metrics, called branch metrics, for each possible transition between states in the code's trellis diagram. The algorithm then uses a dynamic programming approach to find the path through the trellis that minimizes the total branch metric, which corresponds to the most likely sequence of transmitted symbols. Viterbi decoding is a maximum likelihood decoding technique, which means that it maximizes the probability of decoding the correct message given the received sequence of symbols. This makes it a useful tool for decoding convolutional codes in noisy communication channels, where errors are common.

We now describe the Viterbi algorithm according to [15, p. 229]:

Algorithm 1: Viterbi algorithm [15]

```
Initially;
Set encoder to zero state;
Set Viterbi metric zero to initial node;
Set t = 0;
while Not at end of Trellis do
    for each node at time t + 1 do
        Find all predecessors at time t;
        for each predecessor do
            Sum the Viterbi metric of the predecessor and the branch metric
             of the current branch;
        end
        Determine the maximum of the predecessors' branch metrics;
        Set the maximum to current node;
        Label current path with predecessor with shortest path;
    end
    t = t + 1;
end
return the path to terminating node with largest Viterbi metric;
```

2.4 Tail-biting Convolutional Codes

A tail-biting code is created by terminating the convolutional encoder directly after the last input in the input sequence. Tail-biting codes were introduced by Solomon and van Tilborg [18] in 1979 and Ma and Wolf [19] in 1986. This means no rate loss is associated with terminating the convolutional code. However, a consequence is that the last information bits are less protected against channel noise than the other bits. This causes the unequal *error protection property*. This is not desirable in most cases, as we want consistent error protection.

To avoid this problem, you can modify the tail-biting technique by choosing the starting position carefully and then forcing the information sequence to return to that exact origin [2, p. 586]. As codewords without the tail can end in any state, they can start in any state. So, rather than force all states to start in the all-zero state, they can now start in any state. As this state can create the information sequence, we still evade the rate loss discussed earlier in Section 2.3.

2.4.1 Definitions

Let $\mathcal{C}$ be a convolutional code with the rate $R = k/n$ with m memory elements. We obtain the $K^* \times N$ generator matrix G_{TB} as below by terminating the code after K^* information bits [2, p. 591]

$$\mathsf{G}_{\mathrm{TB}} = \begin{bmatrix} \mathsf{G}_0 & \mathsf{G}_1 & \cdots & \mathsf{G}_{m-1} & \mathsf{G}_m & & & & \\ & \mathsf{G}_0 & \mathsf{G}_1 & \cdots & \mathsf{G}_{m-1} & G_m & & & \\ & & \mathsf{G}_0 & \mathsf{G}_1 & \cdots & \mathsf{G}_{m-1} & G_m & & \\ & & & \ddots & \ddots & & & \ddots & \\ & & & & \mathsf{G}_0 & \mathsf{G}_1 & \cdots & & \mathsf{G}_m \end{bmatrix}.$$

The generator of the tail-biting convolutional code is obtained by first removing the last nm columns from G_{TB}, that is, the tail. So that the removed columns are defined by G_{tail} and the remainder of the matrix G_{TB} is $\mathsf{G}_{\text{TB-tail}}$

$$\mathsf{G}_{\text{tail}} = \begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \\ \mathsf{G}_m & & & \\ \mathsf{G}_{m-1} & \mathsf{G}_m & & \\ \vdots & & \ddots & \\ \mathsf{G}_1 & \mathsf{G}_2 & \cdots & \mathsf{G}_m \end{bmatrix},$$

$$\mathsf{G}_{\text{TB-tail}} = \begin{bmatrix} \mathsf{G}_0 & \mathsf{G}_1 & \cdots & \mathsf{G}_{m-1} & \mathsf{G}_m & & & & \\ & \mathsf{G}_0 & \mathsf{G}_1 & \cdots & \mathsf{G}_{m-1} & \mathsf{G}_m & & & \\ & & \mathsf{G}_0 & \mathsf{G}_1 & \cdots & \mathsf{G}_{m-1} & \mathsf{G}_m & & \\ & & & \ddots & \ddots & & & \ddots & \\ & & & & \mathsf{G}_0 & \mathsf{G}_1 & \cdots & & \mathsf{G}_m \\ & & & & & \mathsf{G}_0 & \mathsf{G}_1 & \cdots & \mathsf{G}_{m-1} \\ & & & & & & \ddots & \ddots & \vdots \\ & & & & & & & \ddots & \mathsf{G}_1 \\ & & & & & & & & \mathsf{G}_0 \end{bmatrix}.$$

The $K^* \times N^*$ (where $N^* = N - nm$) tail-biting generator G_{TB} is then made by adding G_{tail} to the first rows of $\mathsf{G}_{\text{TB-tail}}$ and letting it "bite" its own tail. The generator matrix is

$$\mathsf{G}_{\text{TB}} = \begin{bmatrix} \mathsf{G}_0 & \mathsf{G}_1 & \cdots & \mathsf{G}_{m-1} & \mathsf{G}_m & & & & \\ & \mathsf{G}_0 & \mathsf{G}_1 & \cdots & \mathsf{G}_{m-1} & \mathsf{G}_m & & & \\ & & \mathsf{G}_0 & \mathsf{G}_1 & \cdots & \mathsf{G}_{m-1} & \mathsf{G}_m & & \\ & & & \ddots & \ddots & & & \ddots & \\ & & & & \mathsf{G}_0 & \mathsf{G}_1 & \cdots & & \mathsf{G}_m \\ \mathsf{G}_m & & & & & \mathsf{G}_0 & \mathsf{G}_1 & \cdots & \mathsf{G}_{m-1} \\ \mathsf{G}_{m-1} & \mathsf{G}_m & & & & & \ddots & \ddots & \vdots \\ \vdots & & \ddots & & & & & \ddots & \mathsf{G}_1 \\ \mathsf{G}_1 & \mathsf{G}_2 & \cdots & \mathsf{G}_m & & & & & \mathsf{G}_0 \end{bmatrix}.$$

2.5 Lattices

Lattices are a mathematical concept with various applications in science and mathematics, such as crystal structures, coding theory, and cryptography. A lattice Λ is an infinitely extending grid of discrete points in $\mathbb{R}^n$ that repeat periodically, with a constant distance between any two adjacent points, making them uniformly spaced from each other [20].

2.5.1 Definitions

Definition 2.5.1 (Lattice)**.** A generator matrix B for a (full rank) lattice Λ is a matrix whose rows contain a set of n linearly independent basis vectors $\boldsymbol{v_1}, \cdots, \boldsymbol{v_n}$ in $\mathbb{R}^n$. The lattice Λ consists of all possible integer linear combinations of these basis vectors. A lattice is defined as

$$\Lambda = \{a_1\boldsymbol{v_1} + a_2\boldsymbol{v_2} + \cdots + a_n\boldsymbol{v_n} : a_i \in \mathbb{Z}\},$$

where $\boldsymbol{v_1}, \cdots, \boldsymbol{v_n} \in \mathbb{R}^n$ is a set of linearly independent vectors, and $a_1, \cdots, a_n$ are integer coefficients.

A basis for a lattice Λ is not unique, and you can change between basis by using a $n \times m$ change-of-basis matrix U. U is defined as:

- The determinant of the matrix U must be equal to 1 or -1, i.e., $\det(\mathsf{U}) = \pm 1$
- U has only integer entries, i.e.,

$$\mathsf{U} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix},$$

where $a_{ij} \in \mathbb{Z}$, $1 \leq i \leq m, 1 \leq j \leq n$. If $n = m$, then U is a unimodular matrix.

Definition 2.5.2 (Fundamental cell, lattice partition [21, p. 18])**.** The fundamental cell $F_{\boldsymbol{0}}$ of lattice Λ is defined as a bounded set, and when it is shifted by the lattice points it generates what is called a lattice partition, i.e., $F = \{F_{\boldsymbol{\lambda}}\}$ of $\mathbb{R}^n$.

Some properties:

- All cells $F_{\boldsymbol{\lambda}}$ can be seen as $F_{\boldsymbol{0}}$ shifted by a lattice point $\boldsymbol{\lambda} \in \Lambda$, i.e., $F_{\boldsymbol{\lambda}} = F_{\boldsymbol{0}} + \boldsymbol{\lambda}$.
- The lattice partitions do not intersect, that is, $F_{\boldsymbol{\lambda}} \cap F^*_{\boldsymbol{\lambda}} = \emptyset$ for $\boldsymbol{\lambda} \neq \boldsymbol{\lambda}^*$.
- The union of all the lattice partitions covers the whole space $\mathbb{R}$, i.e., $\cup_{\boldsymbol{\lambda}\in\Lambda} F_{\boldsymbol{\lambda}} = \mathbb{R}^n$.

The fundamental cell is often referred to as the fundamental domain.

Definition 2.5.3 (Volume of a Lattice [20, p. 6])**.** The volume of a lattice Λ is

$$\mathrm{vol}\,(\Lambda) = \mathrm{vol}\,(F(\boldsymbol{v_1}, \cdots, \boldsymbol{v_n})) = |\det(\mathsf{B})|\,,$$

where B is the generator matrix of Λ, the volume of Λ is independent of a basis, so it will be the same for all basis.

Definition 2.5.4 (Voronoi region [20, p. 7])**.** The Voronoi region $V_\Lambda(\boldsymbol{\lambda})$ at a point $\boldsymbol{\lambda} \in \Lambda$ is the set of points $\mathbb{R}^n$ which are at least as close to $\boldsymbol{\lambda}$ than to any other lattice points, i.e.

$$V_\Lambda(\boldsymbol{\lambda}) = \{\boldsymbol{x} \in \mathbb{R}^n \colon \|\boldsymbol{\lambda} - \boldsymbol{x}\| \leq \|\boldsymbol{\lambda}^* - \boldsymbol{x}\|, \forall\, \boldsymbol{\lambda}^* \neq \boldsymbol{\lambda} \in \Lambda\}.$$

A lattice's origin is the lattice's zero point, and the Voronoi region at the origin is $V_\Lambda(\boldsymbol{0}) = V(\Lambda)$.

Definition 2.5.5 (Equivalent lattice [21, p. 27])**.** Two lattices Λ_1 and Λ_2 are equivalent, $\Lambda_1 \sim \Lambda_2$, if there exists an orthogonal matrix Q, a unimodular matrix $\tilde{\mathsf{Q}}$ and a a positive scalar, such that $\mathsf{B}_1 = a\mathsf{Q}\mathsf{B}_2\tilde{\mathsf{Q}}$, where B_1 is the generator matrix of Λ_1 and B_2 is the generator matrix of Λ_2.

Definition 2.5.6 (Dual Lattice [20, p. 20])**.** The dual lattice of Λ is the lattice Λ^* defined:

$$\Lambda^* = \{\boldsymbol{y} \in \mathbb{R}^n \colon \langle \boldsymbol{x}, \boldsymbol{y} \rangle \in \mathbb{Z}, \forall\, \boldsymbol{x} \in \Lambda\}.$$

Properties of dual lattices:

- If B is a basis matrix for Λ, then $(\mathsf{B}^\intercal)^{-1}$ is a basis matrix for Λ^*
- $\mathrm{vol}\,(\Lambda^*) = \mathrm{vol}\,(\Lambda)^{-1}$
- If two lattices Λ_1 and Λ_2 is equivalent, $\Lambda_1 \sim \Lambda_2$, then their dual lattices, Λ_1^* and Λ_2^*, are also equivalent $\Lambda_1^* \sim \Lambda_2^*$
- $(\Lambda^*)^* = \Lambda$
- A lattice Λ is said to be unimodular if $\Lambda = \Lambda^*$
- A lattice Λ is said to be isodual if it can be obtained from its dual Λ^* by rotation or reflection.

Definition 2.5.7 (Sublattice [20, p. 17])**.** Let Λ and Λ' be lattices. Λ' is a sublattice of Λ, $\Lambda' \subseteq \Lambda$, if all points in Λ' are also in Λ. Properties of sublattices:

- Let $\mathsf{B_1}$ be a generator matrix for Λ and $\mathsf{B_2}$ a generator matrix for Λ^*. Then $\mathsf{B_2} = \mathsf{A}\mathsf{B_1}$, where A is an integer matrix with $\det(\mathsf{A}) \neq 0$.
- There exists a $\mathcal{B}_1 = \{\boldsymbol{v_1}, \ldots, \boldsymbol{v_n}\}$ for Λ and $\mathcal{B}_2 = \{\boldsymbol{w_1}, \cdots, \boldsymbol{w_n}\}$ for Λ^* such that $\boldsymbol{v_i} = k_i \boldsymbol{w_i}$ where $k_i \in \mathbb{Z}$ for $1 \leq i \leq n$.

2.5.2 Families of Lattices

$\mathbb{Z}^n$ lattices

The cubic lattice, or sometimes called the integer lattice, $\mathbb{Z}^n$ lattice is a subset of $\mathbb{R}^n$, consisting of all points with integer coordinates that can be expressed as linear combinations of n linearly independent basis vectors, also with integer coefficients. More formally, a $\mathbb{Z}^n$ lattice is defined as:

Definition 2.5.8 ($\mathbb{Z}^n$ lattice [20, p. 2])**.** The cubic lattice $\mathbb{Z}^n$ is defined as:

$$\mathbb{Z}^n = \{(x_1, \cdots x_n) \colon x_i \in \mathbb{Z}, \forall\, i = 1, \cdots, n\}.$$

The square lattice $\mathbb{Z}^2$ is perhaps the most well-known lattice. An example of a basis, and maybe the most natural one, is $\{(1, 0), (0, 1)\}$.

Construction A

Construction A lattices [20, p. 37], sometimes called Modulo-q lattices, are constructed from a linear code $\mathscr{C}$ and a cubic lattice scaled to a positive scalar q. Λ_{A} can be described as the result of translating points from $\mathscr{C}$ by points from $q\mathbb{Z}^n$. In our work, $\mathscr{C}$ will be assumed to be a tail-biting convolutional code.

Definition 2.5.9 (Construction A lattice [21, p. 31])**.** For $\mathscr{C}$ a linear code, a Construction A lattice is defined as:

$$\Lambda_{\mathrm{A}}(\mathscr{C}) = q\mathbb{Z}^n + \mathscr{C}.$$

2.5.3 Theta series of a lattice

A Theta series of an integral lattice is a way to describe the lattice. It keeps track of the different norms and the number of lattice points with the specific norms. The theta series is defined as:

$$\Theta_\Lambda(z) = \sum_{\boldsymbol{x}\in\Lambda} N(\boldsymbol{x}) q^{\|\boldsymbol{x}\|^2},$$

where $q = e^{i\pi z}$ for $\mathrm{Im}\,(z) > 0$ and $N(\boldsymbol{x})$ is the number of lattice points at the squared distance $\boldsymbol{x}$ from the origin. The theta series always starts with 1, corresponding to the zero vector [20, p. 53].

Example 4 ($\mathbb{Z}^2$ Lattice)**.** Let us consider the $\mathbb{Z}^2$ lattice in Figure 2.4. The black lattice point is the origin, which is the starting point. Hence,

$$\Theta_{\mathbb{Z}^2}(z) = 1 + \cdots .$$

The second closest lattice points are the red points. There are four so $N(\boldsymbol{x}) = 4$ and they are at distance 1 to the origin, so

$$\Theta_{\mathbb{Z}^2}(z) = 1 + 4q + \cdots ,$$

where $q = e^{\pi i z}$. Next are the green lattice points. There are again four so $N(\boldsymbol{x}) = 4$ and they are at a distance $\sqrt{2}$ away from the origin, which means the next component of $\Theta_L(z)$ is $4q^{(\sqrt{2})^2}$. Continuing like that we obtain the Theta series for $\mathbb{Z}^2$:

$$\Theta_{\mathbb{Z}^2}(z) = 1 + 4q + 4q^2 + 4q^4 + 8q^5 + \cdots$$

Alternatively, some theta series can also be expressed using the Jacobi theta functions, defined as follows.

Definition 2.5.10 (Jacobi theta functions [1, p. 102])**.**

$$\begin{aligned}
\vartheta_2(z) &= \sum_{m\in\mathbb{Z}} q^{(m+\frac{1}{2})^2} = \Theta_{\mathbb{Z}+\frac{1}{2}}(z),\\
\vartheta_3(z) &= \sum_{m\in\mathbb{Z}} q^{m^2} = \Theta_{\mathbb{Z}}(z),\\
\vartheta_4(z) &= \sum_{m\in\mathbb{Z}} (-q)^{m^2}.
\end{aligned}$$

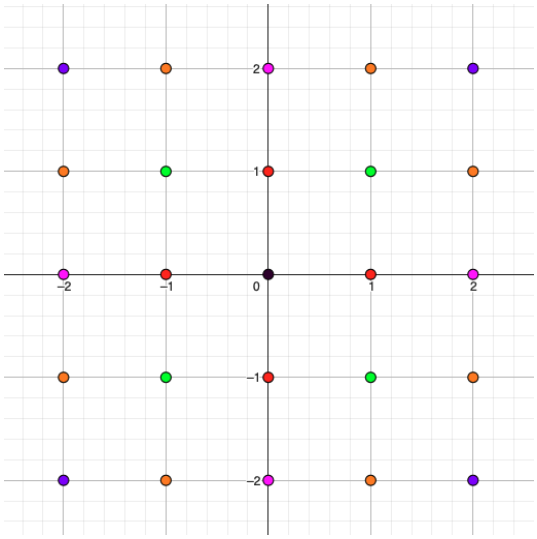

Figure 2.4: $\mathbb{Z}^2$-lattice where all points of the same color have the same distance to the origin.

2.5.4 Formally unimodular lattice and formally self-dual code

Definition 2.5.11 (Formally self-dual code [11, Definition 3])**.** A $[n, k]$ code $\mathscr{C}$ is considered a formally self-dual if and only if

$$W_{\mathscr{C}}(x, y) = \frac{1}{2^k} W_{\mathscr{C}}(x + y, x - y).$$

Definition 2.5.12 (Formally unimodular lattice [11, Definition 11])**.** A lattice Λ is referred to as a formally unimodular lattice, if and only if

$$\Theta_{\Lambda}(z) = \mathrm{vol}\,(\Lambda) \left(\frac{i}{z}\right)^{\frac{n}{2}} \Theta_{\Lambda}\left(-\frac{1}{z}\right).$$

Remark 2. Unimodular and isodual lattices are also formally unimodular lattices.

Lemma 1 ([1, p. 183])**.** The theta series of a Construction A lattice $\Lambda_{\mathrm{A}}(\mathscr{C})$ obtained by a $[n, k]$ code $\mathscr{C}$ with weight enumerator $W_{\mathscr{C}}(x, y)$ is given by

$$\Theta_{\Lambda_{\mathrm{A}}(\mathscr{C})}(z) = W_{\mathscr{C}}(\vartheta_3(2z), \vartheta_2(2z)),$$

where ϑ_2 and ϑ_3 are the Jacobi theta functions given in Definition 2.1.

Remark 3. Lemma 1 implies that if a $[n, n/2]$ code $\mathscr{C}$ is formally self-dual, then the Construction A lattice obtained by this code $\Lambda_{\mathrm{A}}(\mathscr{C})$ is formally self-dual.

Chapter 3

Related work

3.1 Free distance

When discussing good convolutional codes, we often look for the minimum free distance. This is measured in terms of the smallest Hamming weight between all possible codeword pairs of a convolutional code [2, p. 505]. The Hamming weight is defined in Definition 2.1.2. Specifically, the free distance is determined by the weight difference between the all-zero path and the smallest non-zero path through the trellis.

Regarding related results of the free distance of tail-biting codes, in 1993 Brower and Verhoff came out with an updated table for free distance for binary linear codes in [22]. This is what later Bocharova et al. in [23] compared their codes to when they came out with a free distance table for tail-biting codes in 2002.

3.2 Secrecy Gain

When considering lattice coding for the wiretap channel, a notion called secrecy function was proposed to evaluate better the benefit of using a specific lattice Λ_{E} compared to the lattice $\nu\mathbb{Z}^n$, where ν is a scaling factor so that $\mathbb{Z}^n$ is the same volume as Λ_{E}, i.e., $\nu = \mathrm{vol}\,(\Lambda_{\mathrm{E}})^{1/n}$. We define the notion of secrecy gain according to the secrecy function for $\tau > 0$ [20, p. 54].

Definition 3.2.1 (Secrecy function and secrecy gain [7, Definition 3])**.** The secrecy function of a lattice Λ_{E} is

$$\Xi_{\Lambda_{\mathrm{E}}}(\tau) \triangleq \frac{\Theta_{\nu\mathbb{Z}^n}(i\tau)}{\Theta_{\Lambda_{\mathrm{E}}}(i\tau)}.$$

This notion of comparing the secrecy functions tells how much secrecy we can gain by using a lattice Λ_{E} compared to the scaled cubic lattice $\nu\mathbb{Z}^n$. The goal is to determine the global maximum of the secrecy gain, defined as follows.

Definition 3.2.2 (Strong secrecy gain [7, Definition 2])**.** The (strong) secrecy gain of a lattice Λ_{E} is

$$\xi_{\Lambda_{\mathrm{E}}} \triangleq \sup_{\tau>0} \Xi_{\Lambda_{\mathrm{E}}}(\tau).$$

However, this strong secrecy gain is not always easy to calculate, so Belfiore and Solé found in [9] that the secrecy functions of unimodular lattices exhibit a symmetry point, which is what we know as *weak secrecy gain*.

Definition 3.2.3 (Symmetry point)**.** A point $\tau_0 \in \mathbb{R}$ is called a symmetry point if for all $\tau > 0$,

$$\Xi_{\Lambda_E}(\tau_0 \cdot \tau) = \Xi_{\Lambda_E}\left(\frac{\tau_0}{\tau}\right).$$

Definition 3.2.4 (Weak secrecy gain [7, Definition 3])**.** If the secrecy function of a lattice Λ_E has a *symmetry point* τ_0, then the weak secrecy gain corresponds to $\Xi_{\Lambda_E}(\tau_0)$.

In [11], a universal method for determining the strong secrecy gain of a formally unimodular lattice (see Definition 2.5.12) is given, according to the result presented next.

Theorem 1 ([11, Theorem 36])**.** Let $\mathscr{C}$ be a formally self-dual code as defined in Definition 2.5.11. Then

$$[\Xi_{\Lambda_E(\mathscr{C})}(\tau)]^{-1} = \frac{W_{\mathscr{C}}(\sqrt{1+t(\tau)}, \sqrt{1-t(\tau)})}{2^{\frac{n}{2}}} = \frac{f_{\mathscr{C}}(t(\tau))}{2^{\frac{n}{2}}},$$

where $0 < t(\tau) = \vartheta_4^2(i\tau)/\vartheta_3^2(i\tau) < 1$. Moreover, maximizing the secrecy function $\Xi_{\Lambda_E(\mathscr{C})}(\tau)$ is equivalent to determining the minimum of $f_{\mathscr{C}}(t(\tau))$ on $t \in (0,1)$.

Lemma 2 ([11, Lemma 40])**.** If $\mathscr{C}$ is an $[n, 2^{n/2}]$ even formally self-dual code, then we have

$$f_{\mathscr{C}}(t) = 2^{\frac{n}{2}} \sum_{r=0}^{\lfloor \frac{n}{8} \rfloor} a_r (t^4 - t^2 + 1)^r,$$

where $a_r \in \mathbb{Q}$ and $\sum_{r=0}^{\lfloor \frac{n}{8} \rfloor} a_r = 1$

Theorem 2 ([11, Theorem 41])**.** Consider $n \geqslant 8$ and an $[n, 2^{n/2}]$ even formally self-dual code $\mathscr{C}$. If the coefficients a_r of $f_{\mathscr{C}}(t)$ expressed in terms of Lemma 2 satisfy

$$\sum_{r=1}^{\lfloor \frac{n}{8} \rfloor} r a_r \left(\frac{3}{4}\right)^{r-1} > 0,$$

then the secrecy gain of $\Lambda_A(\mathscr{C})$ is achieved at $\tau = 1$.

Proof. We can demonstrate that $f_{\mathscr{C}}(t)$, as defined in Lemma 2 for $0 < t < 1$, attains its minimum value at $t = \frac{1}{\sqrt{2}}$ by showing this is the point where the function changes from decreasing to increasing. To simplify, lets denote $b = \lfloor \frac{n}{8} \rfloor$. Then,

$$\frac{d}{dt} f_{\mathscr{C}}(t) = 2^{\frac{n}{2}} \sum_{r=0}^{b} r a_r h(t)^{r-1} h'(t),$$

Since $h(t) = t^4 - t^2 + 1 = (t^2 - 1/2)^2 + 3/4 \geqslant 3/4$ on $t \in (0,1)$ and $h'(t)$ is not dependent on r, we have

$$\frac{d}{dt} f_{\mathscr{C}}(t) = 2^{\frac{n}{2}} h'(t) \sum_{r=0}^{b} r a_r h(t)^{r-1} \geqslant 2^{\frac{n}{2}} h'(t) \sum_{r=0}^{b} r a_r \left(\frac{3}{4}\right)^{r-1}$$

with $h'(t) = 4t^3 - 2t = 2t(2t^2 - 1)$. The hypothesis being met means that $h'(t)$ has a significant impact on the behavior of the derivative. So it suffices to check the behavior of $h'(t)$

$$h(t) \begin{cases} < 0 & \text{if } 0 < t < \frac{1}{\sqrt{2}} \\ = 0 & \text{if } t = \frac{1}{\sqrt{2}} \\ > 0 & \text{if } \frac{1}{\sqrt{2}} < t < 1 \end{cases}$$

which shows that $f_{\mathscr{C}}(t)$ is indeed decreasing in $t \in (0, 1/\sqrt{2})$ and increasing in $t \in (1/\sqrt{2}, 1)$. □

Remark 4. Since tail-biting convolutional codes are isodual, we can find the secrecy gain of the Construction A lattices created from these codes by applying the results of Theorem 3.2.5 [11].

Chapter 4

Results and Findings

4.1 Implementation

This section will delve into the implementation details of various aspects related to representing trellises, finding which code to use, and calculating their secrecy gain in an efficient way. Specifically, we will cover how the trellis is represented in the code, the process of determining the weight enumerator $W_{\mathscr{C}}(x, y)$ of a code $\mathscr{C}$ using its generator, identifying all codes that have a certain number of memory elements, and subsequently eliminating codes based on different criteria. Finally, we will compare the codes based on their respective secrecy gains.

This is programmed in the programming language Python. Python is a popular language for scientific computing and data analysis, making it well-suited for working with complex mathematical algorithms involved in convolutional codes and lattices. Additionally, Python provides a variety of libraries and frameworks that can be useful for exploring and implementing novel approaches to coding theory. In addition, the ease of learning Python's syntax and the extensive standard library can expedite the development process, freeing up more time to concentrate on the mathematical and theoretical aspects of the research. Most of the utilities required for formula calculation and code reformatting are custom-made for this particular code. Nevertheless, the code also utilizes SymPy [24]'s factor function and some of the built-in functions provided by Python [25].

4.1.1 Trellis

When it came to deciding how to represent the Trellis in the code, several factors had to be taken into consideration. One of the requirements was that given any node, or current state, in the Trellis, the code should be able to determine the next state based on the input and the corresponding output. Upon consideration, it was determined that the most effective method would be to create a dedicated class that would handle these requirements and keep track of the necessary information. Lines 3 through 5 in Algorithm 2 below shows the current state and the next state for inputs 0 and 1.

Calculating the outputs was a more complicated task as this should work for any code of rate $R = 1/2$, for all possible lengths n of the code and constraint length m. This is done by defining a variable memoryInput which is the input inserted at the beginning of the current state and going through both sub generators and XOR-ing

$\oplus$ them with this memoryInput variable.

Example 5. Given the $1/2$ convolutional code $\mathscr{C}$ with $\mathsf{G} = [101, 111]$ and current state 01 the state class will create the memoryInput [001] for the 0 input, and proceed to calculate the output for the 0 input. The first number in the output, or output0 as it is called in Algorithm 2 will be the calculation

$$1 \times 0 \oplus 0 \times 0 \oplus 1 \times 1 = 1$$

and output1 will be

$$1 \times 0 \oplus 1 \times 0 \oplus 1 \times 1 = 1.$$

The output for the 0 input would thus be [11] and for the 1 input it would be [00].

Algorithm 2: Trellis for a convolutional code with generator g

```
   Input: Generator g, CurrentState
 1 class Trellis: ;
 2 procedure Initialize
 3 |   currentState = CurrentState;
 4 |   lastStates = [(currentState - first bit) + [1], (currentState - first bit) +
   |     [0]] ;
 5 |   input0NextState = [0] + (currentState - last bit);
 6 |   input1NextState = [1] + (currentState - last bit);
 7 |   outputForInput0 = calculateOutput(g, 0);
 8 |   outputForInput1 = calculateOutput(g, 1);
 9 procedure calculateOutput(g, input)
10 |   memoryInput = [input] + currentState;
11 |   subgenerator0 = g[0];
12 |   subgenerator1 = g[1];
13 |   output0 = XOR(memoryInput, subgenerator 0);
14 |   output1 = XOR(memoryInput, subgenerator 1);
15 |   return output0, output1;
```

4.1.2 Weight enumerator

The next part of the code is finding the weight enumerator $W_{\mathscr{C}}$ of any given generator using the state class discussed earlier.

Data structure

The first important matter to discuss would be which data structure is needed to keep track of the weight enumerator. Keeping track of the actual codewords was unnecessary as the goal was only to look at the weight enumerator. This simplified the code considerably as there was only a need to store the weight of the codewords, and we could discard the path to the codewords and the codewords themselves. The choice of data structure fell on a simple one-dimensional list with length $n+1$ where n is the code length. For a codeword with weight $w < n$, the number at index w would be augmented by one as there was one more codeword of weight w.

As the programming language follows a 0-index system, the zero codeword of a code $\mathscr{C}$ of length $n = 6$ would be stored in a list with 1 on the 0-index as it has weight $w = 0$, $[1, 0, 0, 0, 0, 0, 0]$. The codeword 111111 would be stored as $[0, 0, 0, 0, 0, 0, 1]$, with a 1 on the index 6, as the weight $w = 6$.

Example 6. Given a code of length 6, the weight list would start as a zero list of length 7

$$\text{weightList} = [0, 0, 0, 0, 0, 0, 0]$$

and given the codeword $c = 111001$, with weight $w = 4$, the weight list would be updated such that at number placed at index 4 will increase with 1, i.e.,

$$\text{weightList} = [0, 0, 0, 0, 1, 0, 0].$$

The algorithm

The first step in the algorithm is to identify all the codewords to then determine the weight enumerator. This requires identifying all possible paths through the trellis while adhering to the tail-biting principle, which forces that the codewords to conclude in the same state as they began. This is done by calculating each state's weight enumerator and merging them to obtain the final result.

To determine all possible weights of codewords in the trellis in an efficient way, we have divided the work up for each time step. At each time t, we go through all states, calculate all weights, and add them to the previous weights. This necessitates storing the weights for the last time step $t - 1$, but not more. For time $t = 0$, we create the dictionary used to keep all the weights for $t - 1$, and it is updated at each time t. A dictionary is used here because we only have to use it to look up specific weights, and dictionaries in Python are extremely fast look-up tables.

In Algorithm 3, at lines 3 through 6, we create the dictionary for the previous weight, and line 10 goes through all states. Subsequently, lines 10 through 30 calculate the new weight for each state at time t, add it to the previous, and when done with all states, set the new weight to be the previous weight and do it all again for all t. Lastly, the weight list for that start state is returned.

This algorithm is then run once for each state, to make it a *tail-biting* convolutional code, and the weight lists for each of the state are added together. After this, the weightList gives us the coefficients of the weight enumerator $W_{\mathscr{C}}$. See Definition 2.1.5 for more information about the weight enumerator.

Example 7. Given the $R = 1/2$, $[n, k] = [8, 4]$ convolutional code $\mathscr{C}$ with $\mathsf{G} = [101, 111]$. The four possible states are $00, 01, 10,$ and 11. To ensure the tail-biting step, the code has to be run once for each of the states, and for each state, we obtain a list with different weights.
In Table 4.1, we see that the weight list for path 00 to 00 has four possibilities, as four weights of codewords are described. One of the codewords is the all-zero codeword with $w = 0$, then two codewords with $w = 5$ and finally one codeword with weight $w = 6$. The final weight list is then the weight list for all four states

$$\begin{aligned} 00&: [1, 0, 0, 0, 0, 2, 1, 0, 0], \\ 01&: [0, 0, 1, 1, 0, 1, 1, 0, 0], \\ 10&: [0, 0, 1, 1, 0, 1, 1, 0, 0], \\ 11&: [0, 0, 0, 2, 1, 0, 1, 0, 0]. \end{aligned}$$

Table 4.1: Weight lists with all states, for a rate $1/2$, $[n, k] = [8, 4]$ tail-biting convolutional code $\mathscr{C}$ with $\mathsf{G} = [101, 111]$

combined, which is $[1, 0, 2, 4, 1, 4, 4, 0, 0]$. From this combined weight list, we obtain the coefficients of our weight enumerator, as from the weight list we can tell that there is one codeword of weight $w = 0$, two codewords of weight $w = 2$, and so on. The complete weight enumerator is:

$$W_{\mathscr{C}}(x, y) = x^8y^0 + 2x^6y^2 + 4x^5y^3 + x^4y^4 + 4x^3y^5 + 4x^2y^6.$$

Algorithm 3: Weight Enumerator of a Convolutional Code

Result: Weight enumerator of a convolutional code with generator g, length n and constraint length μ

Input: g generator, length n, allStates, startState

```
latestWeight = empty list;
for time t = 0 to end of trellis do
    if t = 0 then
        weights = empty list; add zero codeword to weights;
        add [startState, weights] to latestWeight;
    end
    else
        newWeights = empty list;
    end
     for state in allStates do
        weightForState = empty list;
        for previousState in all previous states for state do
            temporaryWeight = empty list;
            if previousState is in latestWeight then
                findOutput = output from previosState to state;
                weight = sum of ones in findOutput;
                if weight ≠ 0 then
                    for latestWeight of state do
                        temporaryWeight[index] = latestWeight + weight;
                    end
                end
                else
                    temporaryWeight[index] = latestWeight;
                end
                add temporaryWeight to weightForState;
            end
            add weightForState to newWeights;
        end
        latestWeights = newWeights;
    end
end
return latestWeights for startState;
```

4.1.3 Making the test data

When creating the test data for the search, the number of test cases was an important element to consider. Before eliminating generators, there are 2^{m+1} possibilities for one generator g where m is the constraint length. That means that for the pair g_1, g_2, there are 2^{2m+2} possibilities to go through.
A few considerations had to be made to ensure that we only tested the necessary generators. We had to ensure that the generators were not catastrophic encoders, that is, make sure that the generators g_1 and g_2 had no common factors. Also, since the weight enumerator $W_{\mathscr{C}}(x, y)$ was the main focus, it was specified that g_1 should be less than g_2, as the weight enumerator would remain the same if $g_1 = a$ and $g_2 = b$ were switched to $g_1 = b$ and $g_2 = a$.
In addition, to remove unnecessary test cases, both g_1 and g_2 also had the constant terms, or the first coefficient, equal to 1 to make sure this is not a non-trivial generator. Below, you can see the number of test cases for each memory element m we tested.

m	Test cases before elimination	Resulting test cases
3	256	27
4	1024	118
5	4096	471
6	16384	1949

Table 4.3: Number of test cases generated for each search for tailbiting convolutional codes, given m

While a lot of test cases were eliminated, the number of tests almost quadruples when increasing m by one. This affects the time complexity a great deal. You can read more about the time complexity in Section 4.2.3.

4.1.4 Calculating the free distance

Calculating the free distance is straightforward with the weight enumerator at hand. As the objective is to identify codewords with minimal weight, excluding the zero codeword, and the weight enumerator $W_{\mathscr{C}}(x, y)$ describes the weight of all codewords in a code $\mathscr{C}$. When creating weight enumerators for the code, the codewords are typically ordered in ascending order of weight. This means that the codeword(s) with the highest weight appear last. So essentially what needs to be done is taking the second element in the weight enumerator, and the free distance will be the weight of that codeword. However, as the weight enumerator is not guaranteed to be in ascending order, the code will go through all parts of the weight enumerator and find the codeword(s) with minimal weight, excluding the zero codeword, and return the free distance.

4.1.5 Calculating the secrecy gain

Calculating the secrecy gain is done by using the formula given in Theorem 1, and $t(\tau)$ is here set to $1/\sqrt{2}$ as concluded in Theorem 2.

Example 8. Consider the rate $R = 1/2$, $[n, k, d] = [14, 7, 4]$, even and self-dual convolutional code $\mathscr{C}$ with weight enumerator

$$W_{\mathscr{C}}(x, y) = x^{14} + 14x^{10}y^4 + 49x^8y^6 + 49x^6y^8 + 14x^4y^{10} + y^{14}.$$

We first check if the conditions of Theorem 2 are satisfied. We find that $a_0 = -3/4$ and $a_1 = 7/4$, and since $\frac{7}{4} \cdot (\frac{3}{4})^0 > 0$, the conditions are being met and the secrecy gain $\Xi_{\Lambda_{\mathrm{E}}(\mathscr{C})}(\tau)$ is achieved at $\tau = 1$.

We then calculate the secrecy gain given the formula in Theorem 1.

$$\left[\Xi_{\Lambda_{\mathrm{E}}(\mathscr{C})}(\tau)\right]^{-1} = \frac{W_{\mathscr{C}}\left(\sqrt{1 + \frac{1}{\sqrt{2}}}, \sqrt{1 - \frac{1}{\sqrt{2}}}\right)}{2^{\frac{14}{2}}} = \underline{\underline{1.778}}.$$

4.2 Results

This section showcases the findings and analysis of our thesis, where we investigate the impact of changing parameters such as constraint length m and code length n on the secrecy gain of a Construction A lattice from a tail-biting convolutional code. We created this lattice using a rate $R = 1/2$ tail-biting convolutional code, and we aim to identify the optimal tail-biting convolutional code, that is, the one that yields the highest secrecy gain. We present our results in searching for this code. Additionally, we explore the $R = 1/2$ tail-biting convolutional code's free distance d_{free} to determine if we can achieve the best free distance for the specific parameters.

4.2.1 Free Distance

To compare and evaluate the method used to compute the weight enumerator and, thereby, the free distance and secrecy gain, we evaluate our findings against the results presented by Bocharova et al. in [23], where the authors investigated the minimum distance for various code lengths n and constraint lengths m. The table below shows the results from this thesis for rate $R = 1/2$ tail-biting convolutional codes for $m = 3$, $m = 5$, and $m = 6$ compared to the rate $R = 1/2$ results from [23] for the same values of m. As Bocharova does not have results for $m = 4$ and $n > 52$ for our other values of m, we have chosen only to include results for $n \leq 52$ for $m = 3$, $m = 5$ and $m = 6$. The rest of the results we found in this search can be found in Appendix A.1.

Free distance comparison						
	$m = 3$		$m = 5$		$m = 6$	
n	Bocharova	This work	Bocharova	This work	Bocharova	This work
12	-	4	-	4	-	4
14	-	4	-	4	-	4
16	5	5	-	5	-	5
18	6	6	-	6	-	6
20	6	6	-	6	-	6
22	-	6	7	7	-	7
24	-	6	-	6	-	8
26	-	6	7	7	-	8
28	-	6	-	7	-	8
30	-	6	8	8	-	8
32	-	6	8	8	-	8
34	-	6	8	8	-	8
36	-	6	8	8	-	8
38	-	6	8	8	-	8
40	-	6	-	8	9	9
42	-	6	-	8	-	9
44	-	6	-	8	10	10
46	-	6	-	8	-	10
48	-	6	-	8	-	9
50	-	6	-	8	10	10
52	-	6	-	8	10	10

Table 4.5: Rate $R = 1/2$ codes; minimum free distance comparison for $n \leq 52$ and $m = 3$, $m = 5$, and $m = 6$.

The results are consistent with the ones from Bocharova et al. This leads us to believe that our methods in evaluating the weight enumerators and finding results are adequate.

4.2.2 Secrecy Gain

In this section, we will look closer at the results obtained while looking at the secrecy gain. The results are obtained using the search method described in Section 4.1, where we look at all possible generators for Construction A lattices created by rate $1/2$ tail-biting convolutional codes for different values of code lengths n and constraint lengths m. The graph below displays our best secrecy gain results for constraint lengths $m = 3$, $m = 4$, $m = 5$, and $m = 6$.

We observe that increasing the constraint length m appears to increase the secrecy gain. This makes sense since a high secrecy gain seems to be correlated with a large minimum distance. The minimum distance of a tail-biting code is upper bounded by the free distance of the convolutional code on which it is based. Moreover, the free distance of the convolutional code is known to increase with a higher trellis complexity, that is, with a larger m. However, for $m = 6$ and code lengths $n < 38$, the secrecy gain has not increased as much as expected from $m = 5$. This can be caused by $n = 38$ not being "long enough". "Long enough" depends on each code and comes from a set of criteria depending on the minimum average cycle weight and how fast the weight grows with the length in the trellis. This is beyond the scope of this thesis, but it can be read about in [26]. We observe that for $m = 6$ and $n \geq 40$, the secrecy gain suddenly becomes significantly larger than for the other constraint lengths, which leads us to believe that the code length $n = 40$ is "long enough".

Also, increasing the code length n increases the difference between the constraint lengths $m = 3$, $m = 4$, $m = 5$, and $m = 6$. At smaller code lengths, such as $n = 12$ to almost $n = 26$, the results for all constraint lengths are either equal or very similar. This is because, for lower code lengths, the lower constraint lengths obtain results that are already very good.

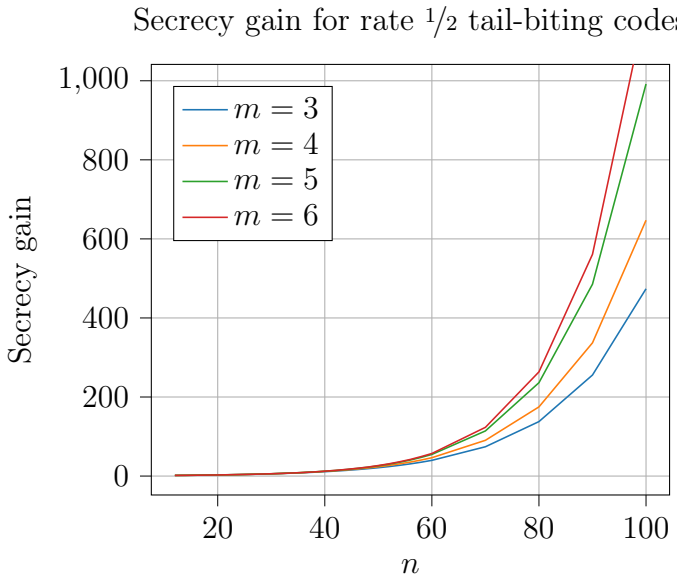

Figure 4.1: Secrecy gain search results for rate $R = 1/2$ tail-biting convolutional codes for $m = 3$, $m = 4$, $m = 5$, and $m = 6$.

The specific values for secrecy gains can be seen in Table 4.7. This table shows the secrecy gains achieved for different code lengths n and constraint lengths m

compared to the upper bound presented in [27], which refers to unimodular lattices only, and also the previously best-known values for secrecy gain $\xi_{\Lambda_A(\mathscr{C})}$ given in [11]. The highlighted values are the best-achieved secrecy gain for each n.

Secrecy Gain comparison						
n	Upper bound [27]	$m=3$	$m=4$	$m=5$	$m=6$	$\xi_{\Lambda_A(\mathscr{C})}$ [11]
12	1.60	1.6569	1.6569	1.6569	1.6569	1.657
14	1.78	1.8283	1.8283	1.8283	1.8283	**1.875**
16	2.21	2.1410	2.1410	2.1410	2.1410	2.141
18	2.49	2.4854	2.4854	2.4854	2.4854	2.485
20	2.81	2.8132	2.8132	**2.8682**	2.8132	**2.868**
22	3.20	3.2426	3.2426	3.3347	3.3347	3.335
24	3.88	3.6741	3.7158	3.7498	**3.8788**	**3.879**
26	4.43	4.3062	4.3062	**4.3555**	**4.3555**	-
28	5.08	4.9093	5.0194	5.0441	**5.0819**	-
30	5.84	5.6848	5.7593	**5.8431**	**5.8431**	**5.843**
32	7.00	6.4898	6.6405	**6.7579**	6.7258	6.748
34	8.06	7.4598	7.7442	**7.8446**	7.7714	-
36	9.31	8.5106	8.8627	**9.0830**	9.0220	-
38	10.77	9.7364	10.2805	**10.5528**	10.440	-
40	12.81	11.0942	11.9029	12.2480	**12.4028**	12.364
42	14.83	12.6526	13.5054	14.1360	14.3805	**14.482**
44	17.20	14.3971	15.5209	16.5191	**16.8268**	-
46	19.98	16.3838	17.8398	19.2623	**19.6065**	-
48	23.66	18.6195	20.5026	22.2955	**22.6636**	-
50	27.48	21.1561	23.5483	25.8932	**26.6343**	-
52	31.96	24.0179	27.0115	30.0883	**31.0259**	-
54	37.21	27.2592	30.9509	34.9928	**36.2326**	-
56	43.93	30.9205	35.4403	40.6476	42,1094	**42.838**
58	51.12	35.0639	40.5623	47.2069	**49.2791**	-
60	59.61	39.7472	46.4075	54.7209	**57.1035**	-
70	130.15	74.1407	90.4576	114.1804	123.2624	**128.368**
80	286.84	137.7886	174.9980	236.1906	**263.5837**	-
90	629.21	255.6120	337.0054	485.3411	**560.9149**	-
100	1383.18	473.7587	647.1535	991.8868	**1191.6391**	-
104	1899.01	606.2983	839.6969	1318.4999	1609.7222	**1885.06**
108	2601.15	775.8770	1089.2789	1751.5857	2173.2995	**2573.53**

Table 4.7: Secrecy gain comparison for $R = 1/2$ tailbiting convolutional codes for constraint lengths $m = 3$ to $m = 6$, for code lengths $n = 12$ to $n = 108$.

When $n < 32$, the table shows that the secrecy gain found for $m = 3$, $m = 4$, $m = 5$, and $m = 6$ are almost equal to or better than the upper bound. It is also worth noting that when $m = 5$ for $n < 38$, the secrecy gain outperforms the outcomes of $m = 3$, $m = 4$, and $m = 6$. However, for $n > 38$ $m = 6$, that exceeds the other values. The results for $m = 6$ improve the secrecy gain for $n = 40$ compared

to the result presented in [11] and otherwise fill in "missing" values.

As n increases, the results become far from the upper bound, and for higher values, the results from [11] are closer to the bound. Even though they still are pretty comparable, this suggests that it may be worthwhile to investigate the performance for larger values of m, such as $m = 7$, which will be discussed more in Section 5.2 Future Work.

The weight enumerators that achieve these secrecy gains can be found in Appendices B.2 to B.5.

4.2.3 Time complexity and secrecy gain for "optimal" convolutional codes

Increasing n when doing this search does not impact the runtime of this code that much. The time complexity is proportional to n^{2^m}; the trellis has 2^m states it has to go through for each n. However, if m increases by 1, the runtime almost triples, and in addition to this, the number of test cases needed to complete the search quadruples. Underneath is two examples of runtimes for $n = 50$ and $n = 100$.

m	one test case (sec)	total test cases	total time (sec)	total time (min)
3	0.06	27	1.58	0.03
4	0.28	118	32.54	0.54
5	0.98	471	462.91	7.72
6	3.85	1949	7501.98	125,03

Table 4.9: Runtime for $n = 50$ search

m	one test case (sec)	total test cases	total time (sec)	total time (min)
3	0.27	27	7.41	0.12
4	0.95	118	111.69	1.86
5	3,04	471	1431.10	23.85
6	13.59	1949	26481.70	441.36

Table 4.11: Runtime for $n = 100$ search

As seen above, while the other factors factor in, the number of test cases is the most impactful factor when it comes to what affects the time of the search.

To test codes for higher values of m without going through all the test cases, we thought it would be useful to examine the potential gains in secrecy provided by rate $1/2$ convolutional codes that are widely regarded as optimal. These convolutional codes are sourced from Lin and Costello's Error Control Coding, as referenced in [2, p. 540]. To compare the secrecy gain correctly, the convolutional codes are first tail-bit, then the secrecy gain is calculated. The results can be seen in Figure 4.2, with the specific secrecy gains attained in Table 4.13.

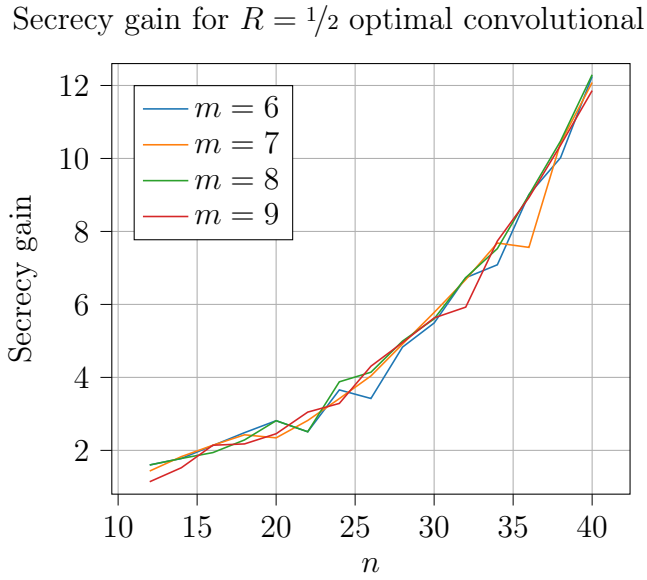

Figure 4.2: Secrecy gain achieved by optimal $R = 1/2$ convolutional codes, for $m \in \{6, \ldots, 9\}$ and $n \in \{12, \ldots, 40\}$.

Secrecy Gain comparison: achieved by optimal convolutional codes [2]					
n	Upper bound [27]	$m = 6$	$m = 7$	$m = 8$	$m = 9$
12	1.60	1.6000	1.4341	1.6000	1.1407
14	1.78	1.7779	1.8283	1.7779	1.5237
16	2.21	2.1333	2.1410	1.9394	2.1410
18	2.49	2.4854	2.4267	2.2858	2.1769
20	2.81	2.8132	2.3431	2.8132	2.4556
22	3.20	2.5099	2.8190	2.5099	3.0484
24	3.88	3.6571	3.4181	**3.8789**	3.2868
26	4.43	3.42110	4.0397	4.1417	4.3062
28	5.08	4.8302	4.9122	4.9951	4.9539
30	5.84	5.4907	5.7787	5.6033	5.6299
32	7.00	6.7313	6.6801	6.7341	5.9256
34	8.06	7.0858	7.6824	7.5286	7.7362
36	9.31	9.01010	7.5630	8.9998	8.9278
38	10.77	10.0193	10.4267	10.4758	10.3616
40	12.81	12.2474	12.0807	12.2940	11.8592
42	14.83	14.2263	14.1914	14.1577	13.9729
44	17.20	16,6562	16.3613	16.4663	16.0636
46	19.98	19.3404	19.5592	19.3322	19.2313
48	23.66	22.6106	22.7140	22.6201	22.4113
50	27.48	26.6343	26.6899	26.6174	26.1239
52	31.96	30.8376	31.1272	31.1427	30.8374
54	37.21	36.2326	36.2983	36.3400	36.4302
56	43.93	42.1094	42.3452	42.5278	42.4884
58	51.12	49.2791	49.8236	49.9045	49.6027
60	59.61	57.1035	58.0952	58.0681	58.3009

Table 4.13: Secrecy gain comparison for rate $R = 1/2$ tail-biting optimal convolutional codes for different code lengths n and constraint lengths m.

Looking at these results, it is interesting to note that an optimal convolutional code does not necessarily translate to optimal secrecy gain for the tail-bit version. Out of all these values, only one $m = 8$, $n = 24$ outperforms both the results from this thesis and previously found in [11]. In addition, the results from this thesis suggest that increasing m should also increase the secrecy gain, and while it increases slightly for some values, there should be a higher increase due to the m. Another interesting thing to note is when looking at $m = 9$, it performs worse or the same as all the lower values of m. As discussed earlier in Section 4.2.2, this may be because the code lengths n used here are not "long enough" for $m = 9$.

4.2.4 Secrecy gain: free distance vs. secrecy gain search

In this section, we evaluate the secrecy gain obtained when specifically searching for secrecy gain rather than for searching distance and then calculate the secrecy gain. All results are obtained using $R = 1/2$ tail-biting convolutional codes. The table below compares the secrecy gain search, here called s, and free distance, here called d_{free}, search results for $m = 3$, $m = 4$, and $m = 5$. The highlighted values are the ones that differ.

We observe that the free distance search also leads to high values for secrecy gain, and most of the time, they are equal to the values obtained by the secrecy gain search. As the minimum free distance is already known, this might mean that if a good secrecy gain suffices, the minimum distance is a good starting point for secrecy gain. Also interesting to note is that for $n \geq 42$, the secrecy gain result for all m seems to coincide with the free distance result. These results differ the most at $n = 20$ and $m = 4$, where the difference is 0.1506. Thus there is no guarantee that the best secrecy gain is achieved if only the free distance is considered, and while the free distance search values are good, the secrecy gain search is always equal or better.

Secrecy gain comparison: Free distance vs. secrecy gain						
	$m = 3$		$m = 4$		$m = 5$	
n	s	d_{free}	s	d_{free}	s	d_{free}
12	**1.6569**	1.6000	1.6569	1.6569	1.6569	1.6569
14	1.8283	1.8283	**1.8283**	1.7778	1.8283	1.8283
16	2.1410	2.1410	2.1410	2.1410	2.1410	2.1410
18	**2.4854**	2.4266	2.4854	2.4854	2.4854	2.4854
20	**2.8132**	2.7078	2.8132	2.8132	2.8682	2.8682
22	3.2426	3.2426	3.2426	3.2426	3.3347	3.3347
24	3.6741	3.6741	**3.7158**	3.6571	3.7498	3.7498
26	4.3062	4.3062	4.3062	4.3062	4.3555	4.3555
28	4.9093	4.9093	5.0194	5.0194	5.0441	5.0441
30	5.6848	5.6848	5.7593	5.7593	5.8431	5.8431
32	6.4898	6.4898	6.6405	6.6405	**6.7569**	6.7341
34	7.4595	7.4595	**7.7442**	7.6368	7.8446	7.8446
36	8.5106	8.5106	8.8627	8.8627	9.0830	9.0830
38	9.7364	9.7364	**10.2805**	10.2196	**10.5528**	10.5262
40	11.0942	11.0942	**11.9029**	11.7523	**12.2480**	12.2131
42	12.6526	12.6526	13.5054	13.5054	14.1360	14.1360
44	14.3971	14.3971	15.5209	15.5209	16.5191	16.5191
46	16.3838	16.3838	17.8398	17.8398	19.2623	19.2326
48	18.6195	18.6195	20.5026	20.5026	22.2955	22.2955
50	21.1561	21.1561	23.5483	23.5483	25.8932	25.8932
52	24.0179	24.0179	27.0115	27.0115	30.0883	30.0883
54	27.2592	27.2592	30.9509	30.9509	34.9928	34.9928
56	30.9205	30.9205	35.4403	35.4403	40.6476	40.6476
58	35.0639	35.0639	40.5623	40.5623	47.2069	47.2069
60	39.7472	39.7472	46.4075	46.4075	54.7209	54.7209

Table 4.15: Secrecy gain comparison for a specific search for secrecy gain vs. a search for distance and then looking at the secrecy. s signifies the secrecy gain search, while d_{free} is the free distance search.

Chapter 5

Conclusions

5.1 Conclusion

Throughout this thesis, we have created and conducted a search to investigate the secrecy gain for Construction A lattices from rate $R = 1/2$ tail-biting convolutional codes and how to maximize it in order to prevent eavesdroppers from guessing a message sent in a wiretap channel. We have seen how different parameters, such as code length and constraint length, impact the secrecy gain and also checked our methods with previous results of minimum free distance.

We have found some improved values for secrecy gain and gotten some good results compared to earlier results and the upper bound from [27]. In addition to this, we have also compared our secrecy gain results to results from optimal convolutional code and concluded that optimal convolutional codes do not mean optimal secrecy gain.

Lastly, we also compared our secrecy gain results to results obtained by looking for the minimum free distance. Then we calculated the secrecy gain and found that the minimum free distance gives some good results. However, it is not guaranteed.

5.2 Future Work

An interesting objective to continue working on would be increasing the code length n and the constraint length m. In this thesis work, some tendencies indicate that increasing m would increase the secrecy gain for the same n and hopefully bring us closer to the upper bounds for longer code lengths. This brings us to the next point: to optimize the code implementation. This could give us results faster and hopefully allow us to increase the parameters and do a more comprehensive search. This can also be done using a more powerful computer or cloud computing.

Another possibility could be to look at different rates for tail-biting convolutional codes. In this thesis only rate $R = 1/2$ was investigated, so $R = 1/3$ and $R = 1/4$ could also be interesting to explore.

Bibliography

[1] J. H. Conway and N. J. A. Sloane, *Sphere Packings, Lattices and Groups*, 3rd ed. New York, NY, USA: Springer, 1999.

[2] S. Lin and D. J. Costello, Jr., *Error Control Coding*, 2nd ed. Upper Saddle River, NJ, USA: Pearson Prentice Hall, 2004.

[3] U. Maurer, "Information-theoretic cryptography," in *Proceedings 19th Annual International Cryptology Conference (CRYPTO)*, Santa Barbara, CA, USA, August 15–19, 1999, pp. 47–65.

[4] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, 3rd ed. New York, NY, USA: Chapman and Hall/CRC, 2020.

[5] D. Micciancio and O. Regev, *Lattice-based Cryptography*. Berlin, Heidelberg, Germany: Springer, 2009, pp. 147–191. [Online]. Available: https://doi.org/10.1007/978-3-540-88702-7_5

[6] J. Hoffstein, J. Pipher, J. H. Silverman, and J. H. Silverman, *An Introduction to Mathematical Cryptography*. New York, NY, USA: Springer, 2008.

[7] F. Oggier, P. Solé, and J.-C. Belfiore, "Lattice codes for the wiretap gaussian channel: Construction and analysis," *IEEE Transactions on Information Theory*, vol. 62, no. 10, pp. 5690–5708, 2016.

[8] J.-C. Belfiore and F. Oggier, "Secrecy gain: A wiretap lattice code design," in *Proceedings IEEE International Symposium on Information Theory and its Applications (ISITA)*, Taichung, Taiwan, October 17–20, 2010, pp. 174–178.

[9] J.-C. Belfiore and P. Solé, "Unimodular lattices for the Gaussian wiretap channel," in *Proceedings IEEE Information Theory Workshop*, Dublin, Ireland, August 30 – September 3, 2010, pp. 1–5.

[10] M. F. Bollauf, H.-Y. Lin, and Ø. Ytrehus, "The secrecy gain of formally unimodular lattices on the Gaussian wiretap channel," in *Proceedings International Zurich Seminar on Information and Communication (IZS)*, Zurich, Switzerland, March 2–4, 2022, pp. 69–73.

[11] ——, "Formally unimodular packings for the Gaussian wiretap channel," June 2022, arXiv:2206.14171v1 [cs.IT].

[12] P. R. Persson, M. F. Bollauf, H.-Y. Lin, and Ø. Ytrehus, "On the secrecy gain of isodual lattices from tail-biting convolutional codes," May 2023, submitted.

[13] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[14] P. Elias, "List decoding for noisy channels," Research Laboratory of Electronics, Massachusetts Institute of Technology (MIT), Tech. Rep., September 20 1957.

[15] R. Johannesson and K. S. Zigangirov, *Fundamentals of Convolutional Coding*, 2nd ed. Hoboken, NJ, USA: John Wiley & Sons, 2015.

[16] G. D. Forney, "The Viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

[17] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[18] G. Solomon and H. C. A. van Tilborg, "A connection between block and convolutional codes," *SIAM Journal on Applied Mathematics*, vol. 37, no. 2, pp. 358–369, 1979.

[19] H. H. Ma and J. K. Wolf, "On tail biting convolutional codes," *IEEE Transactions on Communications*, vol. 34, no. 2, pp. 104–111, 1986.

[20] S. I. R. Costa, F. Oggier, A. Campello, J.-C. Belfiore, and E. Viterbo, *Lattices Applied to Coding for Reliable and Secure Communications*. Cham, Switzerland: Springer, 2017.

[21] R. Zamir, *Lattice Coding for Signals and Networks*. Cambridge, U.K.: Cambridge University Press, 2014.

[22] A. Brouwer and T. Verhoeff, "An updated table of minimum-distance bounds for binary linear codes," *IEEE Transactions on Information Theory*, vol. 39, no. 2, pp. 662–677, 1993.

[23] I. E. Bocharova, R. Johannesson, B. D. Kudryashov, and P. Stahl, "Tailbiting codes: Bounds and search results," *IEEE Transactions on Information Theory*, vol. 48, no. 1, pp. 137–148, 2002.

[24] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, v. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, and A. Scopatz, "SymPy: symbolic computing in Python," p. e103, January 2017. [Online]. Available: https://doi.org/10.7717/peerj-cs.103

[25] Python, "Python," accessed on 10.04.2023. [Online]. Available: https://www.python.org/about/

[26] I. E. Bocharova, M. Handlery, R. Johannesson, and B. D. Kudryashov, "Tailbiting codes obtained via convolutional codes with large active distance-slopes," *IEEE Transactions on Information Theory*, vol. 48, no. 9, pp. 2577–2587, 2002.

[27] F. Lin and F. Oggier, "Gaussian wiretap lattice codes from binary self-dual codes," in *Proceedings IEEE Information Theory Workshop*, Lausanne, Switzerland, September 3–7, 2012, pp. 662–666.

Appendices

Appendix A

Free distance results

A.1 All free distance results

All free distance results				
n	$m = 3$	$m = 4$	$m = 5$	$m = 6$
12	4	4	4	4
14	4	4	4	4
16	5	5	5	5
18	6	6	6	6
20	6	6	6	6
22	6	6	7	7
24	6	6	6	8
26	6	6	7	8
28	6	7	7	8
30	6	7	8	8
32	6	7	8	8
34	6	7	8	8
36	6	7	8	8
38	6	7	8	8
40	6	7	8	9
42	6	7	8	9
44	6	7	8	10
46	6	7	8	10
48	6	7	8	9
50	6	7	8	10
52	6	7	8	10
54	6	7	8	10
56	6	7	8	10
58	6	7	8	10
60	6	7	8	10
70	6	7	8	10
80	6	7	8	10
90	6	7	8	10
100	6	7	8	10

Appendix B

Secrecy gain results

B.1 All secrecy gain results

All secrecy gain results				
n	$m = 3$	$m = 4$	$m = 5$	$m = 6$
12	1.6569	1.6569	1.6569	1.6569
14	1.8283	1.8283	1.8283	1.8283
16	2.1410	2.1410	2.1410	2.1410
18	2.4854	2.4854	2.4854	2.4854
20	2.8132	2.8132	2.8682	2.8132
22	3.2426	3.2426	3.3347	3.3347
24	3.6741	3.7158	3.7498	3.8788
26	4.3062	4.3062	4.3555	4.3555
28	4.9093	5.0194	5.0441	5.0819
30	5.6848	5.7593	5.8431	5.8431
32	6.4898	6.6405	6.7579	6.7258
34	7.4598	7.7442	7.8446	7.7714
36	8.5106	8.8627	9.0830	9.0220
38	9.7364	10.2805	10.5528	10.440
40	11.0942	11.9029	12.2480	12.4028
42	12.6526	13.5054	14.1360	14.3805
44	14.3971	15.5209	16.5191	16.8268
46	16.3838	17.8398	19.2623	19.6065
48	18.6195	20.5026	22.2955	22.6636
50	21.1561	23.5483	25.8932	26.6343
52	24.0179	27.0115	30.0883	31.0259
54	27.2592	30.9509	34.9928	36.2326
56	30.9205	35.4403	40.6476	42.1094
58	35.0639	40.5623	47.2069	49.2791
60	39.7472	46.4075	54.7209	57.1035
70	74.1407	90.4576	114.1804	123.2624
80	137.7886	174.9980	236.1906	263.5837
90	255.6120	337.0054	485.3411	560.9149
100	473.7587	647.1535	991.8868	1191.6391
104	606.2983	839.6969	1318.4999	1609.7222
108	775.8770	1089.2798	1751.5857	2173.2995

B.2 Weight enumerator for constraint length $m = 3$

Secrecy gain: Constraint length 3		
$[n, k, d]$	Secrecy gain	$W_{\mathscr{C}}(x, y)$
$[12, 6, 4]$	1.6569	$x^{12}y^0 + 6x^8y^4 + 24x^7y^5 + 16x^6y^6 + 9x^4y^8 + 8x^3y^9$
$[14, 7, 4]$	1.8283	$x^{14}y^0 + 7x^{10}y^4 + 21x^9y^5 + 21x^8y^6 + 29x^7y^7 + 28x^6y^8 + 7x^5y^9 + 7x^4y^{10} + 7x^3y^{11}$
$[16, 8, 5]$	2.1410	$x^{16}y^0 + 24x^{11}y^5 + 44x^{10}y^6 + 40x^9y^7 + 45x^8y^8 + 40x^7y^9 + 28x^6y^{10} + 24x^5y^{11} + 10x^4y^{12}$
$[18, 9, 6]$	2.4854	$x^{18}y^0 + 102x^{12}y^6 + 153x^{10}y^8 + 153x^8y^{10} + 102x^6y^{12} + 1x^0y^{18}$
$[20, 10, 6]$	2.8132	$x^{20}y^0 + 90x^{14}y^6 + 255x^{12}y^8 + 332x^{10}y^{10} + 255x^8y^{12} + 90x^6y^{14} + 1x^0y^{20}$
$[22, 11, 6]$	3.2426	$x^{22}y^0 + 44x^{16}y^6 + 121x^{15}y^7 + 143x^{14}y^8 + 231x^{13}y^9 + 319x^{12}y^{10} + 298x^{11}y^{11} + 330x^{10}y^{12} + 286x^9y^{13} + 154x^8y^{14} + 77x^7y^{15} + 22x^6y^{16} + 11x^5y^{17} + 11x^4y^{18}$
$[24, 12, 6]$	3.6741	$x^{24}y^0 + 48x^{18}y^6 + 84x^{17}y^7 + 162x^{16}y^8 + 348x^{15}y^9 + 504x^{14}y^{10} + 612x^{13}y^{11} + 568x^{12}y^{12} + 588x^{11}y^{13} + 576x^{10}y^{14} + 316x^9y^{15} + 165x^8y^{16} + 84x^7y^{17} + 24x^6y^{18} + 12x^5y^{19} + 4x^3y^{21}$
$[26, 13, 6]$	4.3062	$x^{26}y^0 + 13x^{20}y^6 + 104x^{19}y^7 + 234x^{18}y^8 + 377x^{17}y^9 + 611x^{16}y^{10} + 910x^{15}y^{11} + 1170x^{14}y^{12} + 1275x^{13}y^{13} + 1235x^{12}y^{14} + 988x^{11}y^{15} + 585x^{10}y^{16} + 351x^9y^{17} + 221x^8y^{18} + 78x^7y^{19} + 26x^6y^{20} + 13x^5y^{21}$
$[28, 14, 6]$	4.9093	$x^{28}y^0 + 14x^{22}y^6 + 100x^{21}y^7 + 175x^{20}y^8 + 378x^{19}y^9 + 903x^{18}y^{10} + 1344x^{17}y^{11} + 1757x^{16}y^{12} + 2254x^{15}y^{13} + 2395x^{14}y^{14} + 2324x^{13}y^{15} + 1988x^{12}y^{16} + 1302x^{11}y^{17} + 749x^{10}y^{18} + 392x^9y^{19} + 175x^8y^{20} + 98x^7y^{21} + 35x^6y^{22}$
$[30, 15, 6]$	5.6848	$x^{30}y^0 + 15x^{24}y^6 + 45x^{23}y^7 + 195x^{22}y^8 + 515x^{21}y^9 + 870x^{20}y^{10} + 1620x^{19}y^{11} + 2630x^{18}y^{12} + 3600x^{17}y^{13} + 4620x^{16}y^{14} + 4734x^{15}y^{15} + 4245x^{14}y^{16} + 3750x^{13}y^{17} + 2650x^{12}y^{18} + 1620x^{11}y^{19} + 978x^{10}y^{20} + 440x^9y^{21} + 165x^8y^{22} + 45x^7y^{23} + 15x^6y^{24} + 15x^5y^{25}$
$[32, 16, 6]$	6.4898	$x^{32}y^0 + 16x^{26}y^6 + 48x^{25}y^7 + 174x^{24}y^8 + 384x^{23}y^9 + 936x^{22}y^{10} + 2096x^{21}y^{11} + 3544x^{20}y^{12} + 5264x^{19}y^{13} + 6984x^{18}y^{14} + 8592x^{17}y^{15} + 9329x^{16}y^{16} + 8592x^{15}y^{17} + 7304x^{14}y^{18} + 5264x^{13}y^{19} + 3292x^{12}y^{20} + 2096x^{11}y^{21} + 1000x^{10}y^{22} + 384x^9y^{23} + 168x^8y^{24} + 48x^7y^{25} + 16x^6y^{26} + 4x^4y^{28}$
$[34, 17, 6]$	7.4595	$x^{34}y^0 + 17x^{28}y^6 + 51x^{27}y^7 + 85x^{26}y^8 + 408x^{25}y^9 + 1105x^{24}y^{10} + 2074x^{23}y^{11} + 4029x^{22}y^{12} + 7276x^{21}y^{13} + 10846x^{20}y^{14} + 14110x^{19}y^{15} + 16779x^{18}y^{16} + 17596x^{17}y^{17} + 16422x^{16}y^{18} + 14348x^{15}y^{19} + 11050x^{14}y^{20} + 7140x^{13}y^{21} + 4097x^{12}y^{22} + 2023x^{11}y^{23} + 935x^{10}y^{24} + 476x^9y^{25} + 153x^8y^{26} + 34x^7y^{27} + 17x^6y^{28}$
$[36, 18, 6]$	8.5106	$x^{36}y^0 + 18x^{30}y^6 + 54x^{29}y^7 + 90x^{28}y^8 + 350x^{27}y^9 + 864x^{26}y^{10} + 2142x^{25}y^{11} + 5037x^{24}y^{12} + 9054x^{23}y^{13} + 14508x^{22}y^{14} + 21132x^{21}y^{15} + 27477x^{20}y^{16} + 33084x^{19}y^{17} + 34792x^{18}y^{18} + 32076x^{17}y^{19} + 28026x^{16}y^{20} + 21804x^{15}y^{21} + 14418x^{14}y^{22} + 8766x^{13}y^{23} + 4752x^{12}y^{24} + 2214x^{11}y^{25} + 936x^{10}y^{26} + 342x^9y^{27} + 153x^8y^{28} + 54x^7y^{29}$
$[38, 19, 6]$	9.7364	$x^{38}y^0 + 19x^{32}y^6 + 57x^{31}y^7 + 95x^{30}y^8 + 209x^{29}y^9 + 874x^{28}y^{10} + 2375x^{27}y^{11} + 4826x^{26}y^{12} + 10260x^{25}y^{13} + 19038x^{24}y^{14} + 29925x^{23}y^{15} + 42826x^{22}y^{16} + 54302x^{21}y^{17} + 62928x^{20}y^{18} + 67242x^{19}y^{19} + 63992x^{18}y^{20} + 55860x^{17}y^{21} + 43339x^{16}y^{22} + 28823x^{15}y^{23} + 18107x^{14}y^{24} + 10393x^{13}y^{25} + 5054x^{12}y^{26} + 2375x^{11}y^{27} + 950x^{10}y^{28} + 304x^9y^{29} + 76x^8y^{30} + 19x^7y^{31} + 19x^6y^{32}$

$[40, 20, 6]$	11.0942	$x^{40}y^0 + 20x^{34}y^6 + 60x^{33}y^7 + 100x^{32}y^8 + 220x^{31}y^9 + 746x^{30}y^{10} + 1900x^{29}y^{11} + 4995x^{28}y^{12} + 11940x^{27}y^{13} + 22760x^{26}y^{14} + 38400x^{25}y^{15} + 59785x^{24}y^{16} + 84940x^{23}y^{17} + 108140x^{22}y^{18} + 124220x^{21}y^{19} + 131046x^{20}y^{20} + 124760x^{19}y^{21} + 108380x^{18}y^{22} + 86220x^{17}y^{23} + 60130x^{16}y^{24} + 37924x^{15}y^{25} + 21850x^{14}y^{26} + 11060x^{13}y^{27} + 5495x^{12}y^{28} + 2340x^{11}y^{29} + 760x^{10}y^{30} + 280x^9y^{31} + 80x^8y^{32} + 20x^7y^{33} + 4x^5y^{35}$
$[42, 21, 6]$	12.6526	$x^{42}y^0 + 21x^{36}y^6 + 63x^{35}y^7 + 105x^{34}y^8 + 231x^{33}y^9 + 525x^{32}y^{10} + 1869x^{31}y^{11} + 5215x^{30}y^{12} + 11550x^{29}y^{13} + 24990x^{28}y^{14} + 48405x^{27}y^{15} + 80724x^{26}y^{16} + 121989x^{25}y^{17} + 168252x^{24}y^{18} + 210483x^{23}y^{19} + 243390x^{22}y^{20} + 258164x^{21}y^{21} + 245553x^{20}y^{22} + 212961x^{19}y^{23} + 169813x^{18}y^{24} + 121233x^{17}y^{25} + 78813x^{16}y^{26} + 47467x^{15}y^{27} + 25203x^{14}y^{28} + 11886x^{13}y^{29} + 5068x^{12}y^{30} + 1995x^{11}y^{31} + 861x^{10}y^{32} + 259x^9y^{33} + 42x^8y^{34} + 21x^7y^{35}$
$[44, 22, 6]$	14.3971	$x^{44}y^0 + 22x^{38}y^6 + 66x^{37}y^7 + 110x^{36}y^8 + 242x^{35}y^9 + 550x^{34}y^{10} + 1608x^{33}y^{11} + 4334x^{32}y^{12} + 11770x^{31}y^{13} + 28105x^{30}y^{14} + 55330x^{29}y^{15} + 99693x^{28}y^{16} + 164604x^{27}y^{17} + 246015x^{26}y^{18} + 337436x^{25}y^{19} + 418605x^{24}y^{20} + 475332x^{23}y^{21} + 500864x^{22}y^{22} + 480150x^{21}y^{23} + 421124x^{20}y^{24} + 339482x^{19}y^{25} + 245476x^{18}y^{26} + 162272x^{17}y^{27} + 99528x^{16}y^{28} + 54274x^{15}y^{29} + 26961x^{14}y^{30} + 12518x^{13}y^{31} + 5104x^{12}y^{32} + 1848x^{11}y^{33} + 583x^{10}y^{34} + 220x^9y^{35} + 77x^8y^{36}$
$[46, 23, 6]$	16.3838	$x^{46}y^0 + 23x^{40}y^6 + 69x^{39}y^7 + 115x^{38}y^8 + 253x^{37}y^9 + 575x^{36}y^{10} + 1265x^{35}y^{11} + 4209x^{34}y^{12} + 11914x^{33}y^{13} + 27071x^{32}y^{14} + 60053x^{31}y^{15} + 120474x^{30}y^{16} + 211692x^{29}y^{17} + 338284x^{28}y^{18} + 494270x^{27}y^{19} + 666724x^{26}y^{20} + 826827x^{25}y^{21} + 937296x^{24}y^{22} + 981434x^{23}y^{23} + 940539x^{22}y^{24} + 824941x^{21}y^{25} + 672359x^{20}y^{26} + 499629x^{19}y^{27} + 336513x^{18}y^{28} + 208104x^{17}y^{29} + 115667x^{16}y^{30} + 59685x^{15}y^{31} + 29095x^{14}y^{32} + 12282x^{13}y^{33} + 4830x^{12}y^{34} + 1748x^{11}y^{35} + 506x^{10}y^{36} + 115x^9y^{37} + 23x^8y^{38} + 23x^7y^{39}$
$[48, 24, 6]$	18.6195	$x^{48}y^0 + 24x^{42}y^6 + 72x^{41}y^7 + 120x^{40}y^8 + 264x^{39}y^9 + 600x^{38}y^{10} + 1320x^{37}y^{11} + 3728x^{36}y^{12} + 10320x^{35}y^{13} + 27396x^{34}y^{14} + 65120x^{33}y^{15} + 134274x^{32}y^{16} + 252984x^{31}y^{17} + 438372x^{30}y^{18} + 693240x^{29}y^{19} + 1002792x^{28}y^{20} + 1329824x^{27}y^{21} + 1624020x^{26}y^{22} + 1834464x^{25}y^{23} + 1920616x^{24}y^{24} + 1849080x^{23}y^{25} + 1632564x^{22}y^{26} + 1336088x^{21}y^{27} + 1003584x^{20}y^{28} + 685776x^{19}y^{29} + 434284x^{18}y^{30} + 251904x^{17}y^{31} + 132741x^{16}y^{32} + 64712x^{15}y^{33} + 28428x^{14}y^{34} + 11976x^{13}y^{35} + 4616x^{12}y^{36} + 1344x^{11}y^{37} + 420x^{10}y^{38} + 120x^9y^{39} + 24x^8y^{40} + 4x^6y^{42}$
$[50, 25, 6]$	21.1561	$x^{50}y^0 + 25x^{44}y^6 + 75x^{43}y^7 + 125x^{42}y^8 + 275x^{41}y^9 + 625x^{40}y^{10} + 1375x^{39}y^{11} + 3225x^{38}y^{12} + 10050x^{37}y^{13} + 27175x^{36}y^{14} + 62955x^{35}y^{15} + 143175x^{34}y^{16} + 296575x^{33}y^{17} + 542650x^{32}y^{18} + 911175x^{31}y^{19} + 1410905x^{30}y^{20} + 2008100x^{29}y^{21} + 2645675x^{28}y^{22} + 3217025x^{27}y^{23} + 3607175x^{26}y^{24} + 3754731x^{25}y^{25} + 3619250x^{24}y^{26} + 3223925x^{23}y^{27} + 2663075x^{22}y^{28} + 2020050x^{21}y^{29} + 1400385x^{20}y^{30} + 900625x^{19}y^{31} + 534500x^{18}y^{32} + 290225x^{17}y^{33} + 147050x^{16}y^{34} + 68005x^{15}y^{35} + 28075x^{14}y^{36} + 10600x^{13}y^{37} + 3700x^{12}y^{38} + 1400x^{11}y^{39} + 400x^{10}y^{40} + 50x^9y^{41} + 25x^8y^{42}$
$[52, 26, 6]$	24.0179	$x^{52}y^0 + 26x^{46}y^6 + 78x^{45}y^7 + 130x^{44}y^8 + 286x^{43}y^9 + 650x^{42}y^{10} + 1430x^{41}y^{11} + 3367x^{40}y^{12} + 9258x^{39}y^{13} + 24362x^{38}y^{14} + 63206x^{37}y^{15} + 151788x^{36}y^{16} + 322556x^{35}y^{17} + 632593x^{34}y^{18} + 1143662x^{33}y^{19} + 1893749x^{32}y^{20} + 2884102x^{31}y^{21} + 4036448x^{30}y^{22} + 5242354x^{29}y^{23} + 6338176x^{28}y^{24} + 7092540x^{27}y^{25} + 7375915x^{26}y^{26} + 7123402x^{25}y^{27} + 6358209x^{24}y^{28} + 5273814x^{23}y^{29} + 4051658x^{22}y^{30} + 2862834x^{21}y^{31} + 1873599x^{20}y^{32} + 1130844x^{19}y^{33} + 627887x^{18}y^{34} + 325962x^{17}y^{35} + 154947x^{16}y^{36} + 67002x^{15}y^{37} + 27378x^{14}y^{38} + 10088x^{13}y^{39} + 3250x^{12}y^{40} + 910x^{11}y^{41} + 299x^{10}y^{42} + 104x^9y^{43}$

$[54, 27, 6]$	27.2592	$x^{54}y^0 + 27x^{48}y^6 + 81x^{47}y^7 + 135x^{46}y^8 + 297x^{45}y^9 + 675x^{44}y^{10} + 1485x^{43}y^{11} + 3510x^{42}y^{12} + 8613x^{41}y^{13} + 23868x^{40}y^{14} + 62118x^{39}y^{15} + 147258x^{38}y^{16} + 340254x^{37}y^{17} + 720621x^{36}y^{18} + 1372194x^{35}y^{19} + 2406348x^{34}y^{20} + 3897846x^{33}y^{21} + 5841207x^{32}y^{22} + 8111232x^{31}y^{23} + 10444734x^{30}y^{24} + 12507696x^{29}y^{25} + 13928895x^{28}y^{26} + 14464516x^{27}y^{27} + 14006088x^{26}y^{28} + 12570336x^{25}y^{29} + 10481085x^{24}y^{30} + 8122410x^{23}y^{31} + 5811669x^{22}y^{32} + 3864258x^{21}y^{33} + 2390175x^{20}y^{34} + 1360638x^{19}y^{35} + 717204x^{18}y^{36} + 348138x^{17}y^{37} + 156762x^{16}y^{38} + 67167x^{15}y^{39} + 25515x^{14}y^{40} + 8775x^{13}y^{41} + 2898x^{12}y^{42} + 783x^{11}y^{43} + 162x^{10}y^{44} + 27x^9y^{45} + 27x^8y^{46}$
$[56, 28, 6]$	30.9205	$x^{56}y^0 + 28x^{50}y^6 + 84x^{49}y^7 + 140x^{48}y^8 + 308x^{47}y^9 + 700x^{46}y^{10} + 1540x^{45}y^{11} + 3654x^{44}y^{12} + 9016x^{43}y^{13} + 22672x^{42}y^{14} + 57288x^{41}y^{15} + 147399x^{40}y^{16} + 354284x^{39}y^{17} + 771442x^{38}y^{18} + 1563912x^{37}y^{19} + 2935051x^{36}y^{20} + 5056436x^{35}y^{21} + 8039556x^{34}y^{22} + 11844476x^{33}y^{23} + 16225125x^{32}y^{24} + 20733608x^{31}y^{25} + 24714550x^{30}y^{26} + 27444088x^{29}y^{27} + 28432727x^{28}y^{28} + 27508796x^{27}y^{29} + 24790976x^{26}y^{30} + 20821500x^{25}y^{31} + 16293564x^{24}y^{32} + 11814880x^{23}y^{33} + 7962542x^{22}y^{34} + 5001000x^{21}y^{35} + 2903369x^{20}y^{36} + 1569036x^{19}y^{37} + 791336x^{18}y^{38} + 366800x^{17}y^{39} + 157451x^{16}y^{40} + 61908x^{15}y^{41} + 23086x^{14}y^{42} + 8148x^{13}y^{43} + 2191x^{12}y^{44} + 588x^{11}y^{45} + 168x^{10}y^{46} + 28x^9y^{47} + 4x^7y^{49}$
$[58, 29, 6]$	35.0639	$x^{58}y^0 + 29x^{52}y^6 + 87x^{51}y^7 + 145x^{50}y^8 + 319x^{49}y^9 + 725x^{48}y^{10} + 1595x^{47}y^{11} + 3799x^{46}y^{12} + 9425x^{45}y^{13} + 22011x^{44}y^{14} + 56579x^{43}y^{15} + 144681x^{42}y^{16} + 345825x^{41}y^{17} + 805620x^{40}y^{18} + 1743625x^{39}y^{19} + 3429047x^{38}y^{20} + 6237494x^{37}y^{21} + 10525144x^{36}y^{22} + 16478293x^{35}y^{23} + 24009535x^{34}y^{24} + 32549194x^{33}y^{25} + 41181218x^{32}y^{26} + 48771823x^{31}y^{27} + 53995013x^{30}y^{28} + 55910812x^{29}y^{29} + 54173392x^{28}y^{30} + 48969893x^{27}y^{31} + 41323144x^{26}y^{32} + 32598378x^{25}y^{33} + 23951564x^{24}y^{34} + 16385899x^{23}y^{35} + 10444901x^{22}y^{36} + 6177986x^{21}y^{37} + 3409211x^{20}y^{38} + 1759720x^{19}y^{39} + 839492x^{18}y^{40} + 373839x^{17}y^{41} + 154889x^{16}y^{42} + 57826x^{15}y^{43} + 19720x^{14}y^{44} + 6235x^{13}y^{45} + 2117x^{12}y^{46} + 580x^{11}y^{47} + 58x^{10}y^{48} + 29x^9y^{49}$
$[60, 30, 6]$	39.7472	$x^{60}y^0 + 30x^{54}y^6 + 90x^{53}y^7 + 150x^{52}y^8 + 330x^{51}y^9 + 750x^{50}y^{10} + 1650x^{49}y^{11} + 3945x^{48}y^{12} + 9840x^{47}y^{13} + 23055x^{46}y^{14} + 55048x^{45}y^{15} + 136965x^{44}y^{16} + 345990x^{43}y^{17} + 829595x^{42}y^{18} + 1843080x^{41}y^{19} + 3840156x^{40}y^{20} + 7421160x^{39}y^{21} + 13240860x^{38}y^{22} + 21918540x^{37}y^{23} + 33728140x^{36}y^{24} + 48460140x^{35}y^{25} + 65141940x^{34}y^{26} + 81887580x^{33}y^{27} + 96415590x^{32}y^{28} + 106333800x^{31}y^{29} + 109902406x^{30}y^{30} + 106584840x^{29}y^{31} + 96746355x^{28}y^{32} + 82129780x^{27}y^{33} + 65291670x^{26}y^{34} + 48444264x^{25}y^{35} + 33532220x^{24}y^{36} + 21716280x^{23}y^{37} + 13116750x^{22}y^{38} + 7389610x^{21}y^{39} + 3886254x^{20}y^{40} + 1899210x^{19}y^{41} + 870310x^{18}y^{42} + 371010x^{17}y^{43} + 144345x^{16}y^{44} + 53000x^{15}y^{45} + 17955x^{14}y^{46} + 5280x^{13}y^{47} + 1335x^{12}y^{48} + 390x^{11}y^{49} + 135x^{10}y^{50}$

$[70, 35, 6]$	74.1407	$x^{70}y^0 + 35x^{64}y^6 + 105x^{63}y^7 + 175x^{62}y^8 + 385x^{61}y^9 + 875x^{60}y^{10} + 1925x^{59}y^{11} + 4690x^{58}y^{12} + 12005x^{57}y^{13} + 28560x^{56}y^{14} + 65590x^{55}y^{15} + 150955x^{54}y^{16} + 344330x^{53}y^{17} + 813540x^{52}y^{18} + 1955520x^{51}y^{19} + 4648637x^{50}y^{20} + 10880945x^{49}y^{21} + 24315375x^{48}y^{22} + 51037280x^{47}y^{23} + 100817080x^{46}y^{24} + 187070296x^{45}y^{25} + 325552570x^{44}y^{26} + 531902665x^{43}y^{27} + 817056035x^{42}y^{28} + 1182157410x^{41}y^{29} + 1613460016x^{40}y^{30} + 2079092400x^{39}y^{31} + 2530822595x^{38}y^{32} + 2912040040x^{37}y^{33} + 3168850580x^{36}y^{34} + 3260920704x^{35}y^{35} + 3173266530x^{34}y^{36} + 2919543410x^{33}y^{37} + 2537227385x^{32}y^{38} + 2082214925x^{31}y^{39} + 1613515043x^{30}y^{40} + 1179373125x^{29}y^{41} + 813081755x^{28}y^{42} + 528624495x^{27}y^{43} + 323708840x^{26}y^{44} + 186904697x^{25}y^{45} + 101737440x^{24}y^{46} + 52102610x^{23}y^{47} + 25155865x^{22}y^{48} + 11418790x^{21}y^{49} + 4849712x^{20}y^{50} + 1927520x^{19}y^{51} + 716065x^{18}y^{52} + 253085x^{17}y^{53} + 82005x^{16}y^{54} + 23282x^{15}y^{55} + 6510x^{14}y^{56} + 1610x^{13}y^{57} + 280x^{12}y^{58} + 35x^{11}y^{59} + 35x^{10}y^{60}$
$[80, 40, 6]$	137.7886	$x^{80}y^0 + 40x^{74}y^6 + 120x^{73}y^7 + 200x^{72}y^8 + 440x^{71}y^9 + 1000x^{70}y^{10} + 2200x^{69}y^{11} + 5460x^{68}y^{12} + 14320x^{67}y^{13} + 34540x^{66}y^{14} + 80160x^{65}y^{15} + 186620x^{64}y^{16} + 430520x^{63}y^{17} + 986180x^{62}y^{18} + 2258240x^{61}y^{19} + 5174514x^{60}y^{20} + 11941000x^{59}y^{21} + 27838340x^{58}y^{22} + 64519320x^{57}y^{23} + 146644435x^{56}y^{24} + 323665008x^{55}y^{25} + 684312340x^{54}y^{26} + 1374809240x^{53}y^{27} + 2618607160x^{52}y^{28} + 4722678320x^{51}y^{29} + 8063237864x^{50}y^{30} + 13039264600x^{49}y^{31} + 19985492585x^{48}y^{32} + 29064844280x^{47}y^{33} + 40147624180x^{46}y^{34} + 52711216616x^{45}y^{35} + 65822953860x^{44}y^{36} + 78214698480x^{43}y^{37} + 88464750900x^{42}y^{38} + 95267371400x^{41}y^{39} + 97686380174x^{40}y^{40} + 95361896440x^{39}y^{41} + 88615254080x^{38}y^{42} + 78362271080x^{37}y^{43} + 65917026300x^{36}y^{44} + 52733314096x^{35}y^{45} + 40105839980x^{34}y^{46} + 28984134960x^{33}y^{47} + 19902092090x^{32}y^{48} + 12980691080x^{31}y^{49} + 8038717308x^{30}y^{50} + 4727617200x^{29}y^{51} + 2639516690x^{28}y^{52} + 1398195400x^{27}y^{53} + 702531260x^{26}y^{54} + 334412968x^{25}y^{55} + 150797495x^{24}y^{56} + 64358560x^{23}y^{57} + 25891620x^{22}y^{58} + 9843240x^{21}y^{59} + 3523528x^{20}y^{60} + 1172080x^{19}y^{61} + 361760x^{18}y^{62} + 105240x^{17}y^{63} + 30080x^{16}y^{64} + 6920x^{15}y^{65} + 1260x^{14}y^{66} + 360x^{13}y^{67} + 40x^{12}y^{68} + 4x^{10}y^{70}$

$[90, 45, 6]$	255.6120	$x^{90}y^0 + 45x^{84}y^6 + 135x^{83}y^7 + 225x^{82}y^8 + 495x^{81}y^9 + 1125x^{80}y^{10} + 2475x^{79}y^{11} + 6255x^{78}y^{12} + 16785x^{77}y^{13} + 40995x^{76}y^{14} + 96030x^{75}y^{15} + 225810x^{74}y^{16} + 525960x^{73}y^{17} + 1217715x^{72}y^{18} + 2825145x^{71}y^{19} + 6528105x^{70}y^{20} + 14938200x^{69}y^{21} + 33903405x^{68}y^{22} + 76878675x^{67}y^{23} + 174553755x^{66}y^{24} + 396709128x^{65}y^{25} + 899715915x^{64}y^{26} + 2012704895x^{63}y^{27} + 4391555670x^{62}y^{28} + 9273421800x^{61}y^{29} + 18817613340x^{60}y^{30} + 36517775490x^{59}y^{31} + 67620538755x^{58}y^{32} + 119362144500x^{57}y^{33} + 200813340420x^{56}y^{34} + 322076133387x^{55}y^{35} + 492686229080x^{54}y^{36} + 719307622440x^{53}y^{37} + 1002956573430x^{52}y^{38} + 1336407443325x^{51}y^{39} + 1702597236606x^{50}y^{40} + 2074817468250x^{49}y^{41} + 2419289316660x^{48}y^{42} + 2699825319315x^{47}y^{43} + 2883888642285x^{46}y^{44} + 2948750720586x^{45}y^{45} + 2885933674395x^{44}y^{46} + 2703117454650x^{43}y^{47} + 2422741124220x^{42}y^{48} + 2077353320400x^{41}y^{49} + 1703577626955x^{40}y^{50} + 1335865623495x^{39}y^{51} + 1001353404465x^{38}y^{52} + 717337670760x^{37}y^{53} + 491025632485x^{36}y^{54} + 321095575737x^{35}y^{55} + 200554545195x^{34}y^{56} + 119629086120x^{33}y^{57} + 68125530015x^{32}y^{58} + 37030054905x^{31}y^{59} + 19206443940x^{30}y^{60} + 9499467600x^{29}y^{61} + 4478776470x^{28}y^{62} + 2011463270x^{27}y^{63} + 859152510x^{26}y^{64} + 348611508x^{25}y^{65} + 134188590x^{24}y^{66} + 48997845x^{23}y^{67} + 16906590x^{22}y^{68} + 5463000x^{21}y^{69} + 1668699x^{20}y^{70} + 477360x^{19}y^{71} + 122955x^{18}y^{72} + 28935x^{17}y^{73} + 7245x^{16}y^{74} + 1770x^{15}y^{75} + 90x^{14}y^{76} + 45x^{13}y^{77}$
$[100, 50, 6]$	473.7587	$x^{100}y^0 + 50x^{94}y^6 + 150x^{93}y^7 + 250x^{92}y^8 + 550x^{91}y^9 + 1250x^{90}y^{10} + 2750x^{89}y^{11} + 7075x^{88}y^{12} + 19400x^{87}y^{13} + 47925x^{86}y^{14} + 113200x^{85}y^{15} + 268525x^{84}y^{16} + 630650x^{83}y^{17} + 1473725x^{82}y^{18} + 3459050x^{81}y^{19} + 8094075x^{80}y^{20} + 18757000x^{79}y^{21} + 43131825x^{78}y^{22} + 98529200x^{77}y^{23} + 223462475x^{76}y^{24} + 503633672x^{75}y^{25} + 1131522550x^{74}y^{26} + 2539624750x^{73}y^{27} + 5686347425x^{72}y^{28} + 12665629200x^{71}y^{29} + 27911967335x^{70}y^{30} + 60355979750x^{69}y^{31} + 127097637025x^{68}y^{32} + 259133952300x^{67}y^{33} + 509205454275x^{66}y^{34} + 961396177160x^{65}y^{35} + 1740981156500x^{64}y^{36} + 3021331048550x^{63}y^{37} + 5023550861825x^{62}y^{38} + 8003538336400x^{61}y^{39} + 12222135287495x^{60}y^{40} + 17897516181700x^{59}y^{41} + 25143462105100x^{58}y^{42} + 33903537094350x^{57}y^{43} + 43897034661825x^{56}y^{44} + 54594893877180x^{55}y^{45} + 65241345009725x^{54}y^{46} + 74928519088050x^{53}y^{47} + 82717037539225x^{52}y^{48} + 87781846540800x^{51}y^{49} + 89554479721091x^{50}y^{50} + 87827333651000x^{49}y^{51} + 82793074802600x^{48}y^{52} + 75011237349850x^{47}y^{53} + 65307117803075x^{46}y^{54} + 54627954764760x^{45}y^{55} + 43893929334825x^{44}y^{56} + 33872298777000x^{43}y^{57} + 25098418933150x^{42}y^{58} + 17853825510050x^{41}y^{59} + 12190722307935x^{40}y^{60} + 7988464745400x^{39}y^{61} + 5023033688925x^{38}y^{62} + 3030164934250x^{37}y^{63} + 1753348775250x^{36}y^{64} + 972942607220x^{35}y^{65} + 517616886925x^{34}y^{66} + 263914519600x^{33}y^{67} + 128904866600x^{32}y^{68} + 60278465050x^{31}y^{69} + 26968966105x^{30}y^{70} + 11535762450x^{29}y^{71} + 4711374475x^{28}y^{72} + 1835755750x^{27}y^{73} + 681486750x^{26}y^{74} + 240248420x^{25}y^{75} + 80237500x^{24}y^{76} + 25379700x^{23}y^{77} + 7618250x^{22}y^{78} + 2132750x^{21}y^{79} + 548550x^{20}y^{80} + 136550x^{19}y^{81} + 30800x^{18}y^{82} + 5350x^{17}y^{83} + 1025x^{16}y^{84} + 350x^{15}y^{85}$

$[104, 52, 6]$	606.2983	$x^{104}y^0+52x^{98}y^6+156x^{97}y^7+260x^{96}y^8+572x^{95}y^9+1300x^{94}y^{10}+2860x^{93}y^{11}+7410x^{92}y^{12}+20488x^{91}y^{13}+50830x^{90}y^{14}+120432x^{89}y^{15}+286598x^{88}y^{16}+675116x^{87}y^{17}+1583010x^{86}y^{18}+3731624x^{85}y^{19}+8772634x^{84}y^{20}+20423624x^{83}y^{21}+47187634x^{82}y^{22}+108329156x^{81}y^{23}+246965095x^{80}y^{24}+559313872x^{79}y^{25}+1259128578x^{78}y^{26}+2822174992x^{77}y^{27}+6309038502x^{76}y^{28}+14062641632x^{75}y^{29}+31195323250x^{74}y^{30}+68574083656x^{73}y^{31}+148343599092x^{72}y^{32}+313632505524x^{71}y^{33}+644302005308x^{70}y^{34}+1279926048388x^{69}y^{35}+2450259752577x^{68}y^{36}+4510645690252x^{67}y^{37}+7975137654614x^{66}y^{38}+13535537123420x^{65}y^{39}+22049601250157x^{64}y^{40}+34480904427060x^{63}y^{41}+51778256188230x^{62}y^{42}+74693055380580x^{61}y^{43}+103552240451972x^{60}y^{44}+138025790338280x^{59}y^{45}+176945672262294x^{58}y^{46}+218240784477664x^{57}y^{47}+259036585155920x^{56}y^{48}+295938759563356x^{55}y^{49}+325475846331428x^{54}y^{50}+344623983422628x^{53}y^{51}+351310678844451x^{52}y^{52}+344782754322084x^{51}y^{53}+325744214755050x^{50}y^{54}+296235464504148x^{49}y^{55}+259278954302641x^{48}y^{56}+218371435132396x^{47}y^{57}+176947959279806x^{46}y^{58}+137924115594460x^{45}y^{59}+103395270390126x^{44}y^{60}+74533452115824x^{43}y^{61}+51656381407046x^{42}y^{62}+34414996497960x^{41}y^{63}+22037102178871x^{40}y^{64}+13560677591036x^{39}y^{65}+8017723936508x^{38}y^{66}+4553860207884x^{37}y^{67}+2484089932207x^{36}y^{68}+1300996127028x^{35}y^{69}+653971561438x^{34}y^{70}+315368756456x^{33}y^{71}+145814837415x^{32}y^{72}+64605679008x^{31}y^{73}+27405789606x^{30}y^{74}+11117465216x^{29}y^{75}+4307770454x^{28}y^{76}+1592329024x^{27}y^{77}+560896804x^{26}y^{78}+187605184x^{25}y^{79}+59429214x^{24}y^{80}+17867720x^{23}y^{81}+5037890x^{22}y^{82}+1313572x^{21}y^{83}+321971x^{20}y^{84}+78676x^{19}y^{85}+16380x^{18}y^{86}+2184x^{17}y^{87}+624x^{16}y^{88}+52x^{15}y^{89}+4x^{13}y^{91}$

$[108, 54, 6]$	775.8770	$x^{108}y^0 + 54x^{102}y^6 + 162x^{101}y^7 + 270x^{100}y^8 + 594x^{99}y^9 + 1350x^{98}y^{10} + 2970x^{97}y^{11} + 7749x^{96}y^{12} + 21600x^{95}y^{13} + 53811x^{94}y^{14} + 127872x^{93}y^{15} + 305235x^{92}y^{16} + 721062x^{91}y^{17} + 1696239x^{90}y^{18} + 4015170x^{89}y^{19} + 9481509x^{88}y^{20} + 22171536x^{87}y^{21} + 51457491x^{86}y^{22} + 118685412x^{85}y^{23} + 271896300x^{84}y^{24} + 618970302x^{83}y^{25} + 1400571621x^{82}y^{26} + 3149004410x^{81}y^{27} + 7042877856x^{80}y^{28} + 15694576038x^{79}y^{29} + 34850615580x^{78}y^{30} + 77023895598x^{77}y^{31} + 168878508984x^{76}y^{32} + 365240357910x^{75}y^{33} + 774382533606x^{74}y^{34} + 1600508453598x^{73}y^{35} + 3208949737953x^{72}y^{36} + 6218132187510x^{71}y^{37} + 11616240101868x^{70}y^{38} + 20888482023378x^{69}y^{39} + 36125926872984x^{68}y^{40} + 60068803234050x^{67}y^{41} + 96025169690190x^{66}y^{42} + 147607824259026x^{65}y^{43} + 218250798727329x^{64}y^{44} + 310513767066846x^{63}y^{45} + 425251109899746x^{62}y^{46} + 560793771336726x^{61}y^{47} + 712349832351528x^{60}y^{48} + 871840276181802x^{59}y^{49} + 1028331263962764x^{58}y^{50} + 1169114528292282x^{57}y^{51} + 1281331641725667x^{56}y^{52} + 1353869116113006x^{55}y^{53} + 1379149703906902x^{54}y^{54} + 1354429207485630x^{53}y^{55} + 1282284459224640x^{52}y^{56} + 1170181204074018x^{51}y^{57} + 1029222710526996x^{50}y^{58} + 872349570935946x^{49}y^{59} + 712405877287311x^{48}y^{60} + 560467262160306x^{47}y^{61} + 424704971608668x^{46}y^{62} + 309932502698922x^{45}y^{63} + 217782830348631x^{44}y^{64} + 147330345081906x^{43}y^{65} + 95941993457754x^{42}y^{66} + 60132984895818x^{41}y^{67} + 36268961749011x^{40}y^{68} + 21047387552226x^{39}y^{69} + 11749434006582x^{38}y^{70} + 6307810667316x^{37}y^{71} + 3255785400258x^{36}y^{72} + 1615104374148x^{35}y^{73} + 769673426076x^{34}y^{74} + 352154946924x^{33}y^{75} + 154592263638x^{32}y^{76} + 65061753066x^{31}y^{77} + 26228369907x^{30}y^{78} + 10114269714x^{29}y^{79} + 3726611253x^{28}y^{80} + 1310351976x^{27}y^{81} + 438283845x^{26}y^{82} + 139026780x^{25}y^{83} + 41806980x^{24}y^{84} + 11944746x^{23}y^{85} + 3192831x^{22}y^{86} + 781110x^{21}y^{87} + 185004x^{20}y^{88} + 40392x^{19}y^{89} + 6615x^{18}y^{90} + 1188x^{17}y^{91} + 405x^{16}y^{92}$

B.3 Weight enumerator for constraint length $m = 4$

Secrecy gain: Constraint length 4		
$[n, k, d]$	Secrecy gain	$W_{\mathscr{C}}(x, y)$
$[12, 6, 4]$	1.6569	$x^{12}y^0 + 6x^8y^4 + 24x^7y^5 + 16x^6y^6 + 9x^4y^8 + 8x^3y^9$
$[14, 7, 4]$	1.8283	$x^{14}y^0 + 7x^{10}y^4 + 21x^9y^5 + 21x^8y^6 + 29x^7y^7 + 28x^6y^8 + 7x^5y^9 + 7x^4y^{10} + 7x^3y^{11}$
$[16, 8, 5]$	2.1410	$x^{16}y^0 + 24x^{11}y^5 + 44x^{10}y^6 + 40x^9y^7 + 45x^8y^8 + 40x^7y^9 + 28x^6y^{10} + 24x^5y^{11} + 10x^4y^{12}$
$[18, 9, 6]$	2.4854	$x^{18}y^0 + 102x^{12}y^6 + 153x^{10}y^8 + 153x^8y^{10} + 102x^6y^{12} + 1x^0y^{18}$
$[20, 10, 6]$	2.8132	$x^{20}y^0 + 90x^{14}y^6 + 255x^{12}y^8 + 332x^{10}y^{10} + 255x^8y^{12} + 90x^6y^{14} + 1x^0y^{20}$
$[22, 11, 6]$	3.2426	$x^{22}y^0 + 44x^{16}y^6 + 121x^{15}y^7 + 143x^{14}y^8 + 231x^{13}y^9 + 319x^{12}y^{10} + 298x^{11}y^{11} + 330x^{10}y^{12} + 286x^9y^{13} + 154x^8y^{14} + 77x^7y^{15} + 22x^6y^{16} + 11x^5y^{17} + 11x^4y^{18}$
$[24, 12, 6]$	3.7158	$x^{24}y^0 + 30x^{18}y^6 + 120x^{17}y^7 + 174x^{16}y^8 + 304x^{15}y^9 + 522x^{14}y^{10} + 552x^{13}y^{11} + 608x^{12}y^{12} + 672x^{11}y^{13} + 498x^{10}y^{14} + 328x^9y^{15} + 177x^8y^{16} + 48x^7y^{17} + 38x^6y^{18} + 24x^5y^{19}$
$[26, 13, 6]$	4.3062	$x^{26}y^0 + 13x^{20}y^6 + 104x^{19}y^7 + 234x^{18}y^8 + 377x^{17}y^9 + 611x^{16}y^{10} + 910x^{15}y^{11} + 1170x^{14}y^{12} + 1275x^{13}y^{13} + 1235x^{12}y^{14} + 988x^{11}y^{15} + 585x^{10}y^{16} + 351x^9y^{17} + 221x^8y^{18} + 78x^7y^{19} + 26x^6y^{20} + 13x^5y^{21}$
$[28, 14, 7]$	5.0194	$x^{28}y^0 + 72x^{21}y^7 + 301x^{20}y^8 + 392x^{19}y^9 + 672x^{18}y^{10} + 1512x^{17}y^{11} + 1736x^{16}y^{12} + 1960x^{15}y^{13} + 2752x^{14}y^{14} + 2520x^{13}y^{15} + 1771x^{12}y^{16} + 1176x^{11}y^{17} + 672x^{10}y^{18} + 504x^9y^{19} + 280x^8y^{20} + 56x^7y^{21} + 7x^4y^{24}$
$[30, 15, 7]$	5.7593	$x^{30}y^0 + 75x^{23}y^7 + 195x^{22}y^8 + 440x^{21}y^9 + 990x^{20}y^{10} + 1620x^{19}y^{11} + 2510x^{18}y^{12} + 3720x^{17}y^{13} + 4470x^{16}y^{14} + 4674x^{15}y^{15} + 4485x^{14}y^{16} + 3720x^{13}y^{17} + 2650x^{12}y^{18} + 1620x^{11}y^{19} + 858x^{10}y^{20} + 440x^9y^{21} + 210x^8y^{22} + 75x^7y^{23} + 15x^6y^{24}$
$[32, 16, 7]$	6.6405	$x^{32}y^0 + 64x^{25}y^7 + 152x^{24}y^8 + 416x^{23}y^9 + 1104x^{22}y^{10} + 1936x^{21}y^{11} + 3276x^{20}y^{12} + 5424x^{19}y^{13} + 7256x^{18}y^{14} + 8464x^{17}y^{15} + 9181x^{16}y^{16} + 8688x^{15}y^{17} + 7208x^{14}y^{18} + 5424x^{13}y^{19} + 3452x^{12}y^{20} + 1808x^{11}y^{21} + 936x^{10}y^{22} + 496x^9y^{23} + 194x^8y^{24} + 48x^7y^{25} + 8x^6y^{26}$
$[34, 17, 7]$	7.7442	$x^{34}y^0 + 34x^{27}y^7 + 119x^{26}y^8 + 476x^{25}y^9 + 1071x^{24}y^{10} + 2023x^{23}y^{11} + 4284x^{22}y^{12} + 7140x^{21}y^{13} + 10200x^{20}y^{14} + 14348x^{19}y^{15} + 17187x^{18}y^{16} + 17596x^{17}y^{17} + 16830x^{16}y^{18} + 14110x^{15}y^{19} + 10404x^{14}y^{20} + 7276x^{13}y^{21} + 4352x^{12}y^{22} + 2074x^{11}y^{23} + 901x^{10}y^{24} + 408x^9y^{25} + 187x^8y^{26} + 51x^7y^{27}$
$[36, 18, 7]$	8.8627	$x^{36}y^0 + 54x^{29}y^7 + 81x^{28}y^8 + 294x^{27}y^9 + 1161x^{26}y^{10} + 2268x^{25}y^{11} + 4434x^{24}y^{12} + 9144x^{23}y^{13} + 14661x^{22}y^{14} + 20868x^{21}y^{15} + 28251x^{20}y^{16} + 32724x^{19}y^{17} + 33874x^{18}y^{18} + 32976x^{17}y^{19} + 28260x^{16}y^{20} + 21528x^{15}y^{21} + 14706x^{14}y^{22} + 8406x^{13}y^{23} + 4419x^{12}y^{24} + 2358x^{11}y^{25} + 1125x^{10}y^{26} + 452x^9y^{27} + 90x^8y^{28} + 9x^6y^{30}$
$[38, 19, 7]$	10.2805	$x^{38}y^0 + 38x^{31}y^7 + 95x^{30}y^8 + 266x^{29}y^9 + 893x^{28}y^{10} + 2299x^{27}y^{11} + 5073x^{26}y^{12} + 10583x^{25}y^{13} + 18848x^{24}y^{14} + 28918x^{23}y^{15} + 42123x^{22}y^{16} + 55480x^{21}y^{17} + 63574x^{20}y^{18} + 67242x^{19}y^{19} + 64638x^{18}y^{20} + 54682x^{17}y^{21} + 42636x^{16}y^{22} + 29830x^{15}y^{23} + 17917x^{14}y^{24} + 10070x^{13}y^{25} + 5301x^{12}y^{26} + 2451x^{11}y^{27} + 969x^{10}y^{28} + 247x^9y^{29} + 76x^8y^{30} + 38x^7y^{31}$
$[40, 20, 7]$	11.9029	$x^{40}y^0 + 40x^{33}y^7 + 70x^{32}y^8 + 180x^{31}y^9 + 900x^{30}y^{10} + 2200x^{29}y^{11} + 4985x^{28}y^{12} + 11800x^{27}y^{13} + 22460x^{26}y^{14} + 37964x^{25}y^{15} + 60265x^{24}y^{16} + 84920x^{23}y^{17} + 107570x^{22}y^{18} + 124640x^{21}y^{19} + 131330x^{20}y^{20} + 125440x^{19}y^{21} + 108400x^{18}y^{22} + 85520x^{17}y^{23} + 60340x^{16}y^{24} + 37364x^{15}y^{25} + 21520x^{14}y^{26} + 11560x^{13}y^{27} + 5605x^{12}y^{28} + 2440x^{11}y^{29} + 772x^{10}y^{30} + 220x^9y^{31} + 60x^8y^{32} + 10x^6y^{34}$

$[42, 21, 7]$	13.5054	$x^{42}y^0 + 63x^{35}y^7 + 63x^{34}y^8 + 133x^{33}y^9 + 819x^{32}y^{10} + 1827x^{31}y^{11} + 4606x^{30}y^{12} + 12789x^{29}y^{13} + 25998x^{28}y^{14} + 46228x^{27}y^{15} + 80199x^{26}y^{16} + 122472x^{25}y^{17} + 166761x^{24}y^{18} + 212982x^{23}y^{19} + 245364x^{22}y^{20} + 254426x^{21}y^{21} + 245364x^{20}y^{22} + 214935x^{19}y^{23} + 169729x^{18}y^{24} + 122661x^{17}y^{25} + 78309x^{16}y^{26} + 44919x^{15}y^{27} + 24846x^{14}y^{28} + 12705x^{13}y^{29} + 5950x^{12}y^{30} + 2310x^{11}y^{31} + 504x^{10}y^{32} + 126x^9y^{33} + 63x^8y^{34}$
$[44, 22, 7]$	15.5209	$x^{44}y^0 + 66x^{37}y^7 + 66x^{36}y^8 + 110x^{35}y^9 + 748x^{34}y^{10} + 1584x^{33}y^{11} + 4136x^{32}y^{12} + 12870x^{31}y^{13} + 28149x^{30}y^{14} + 53834x^{29}y^{15} + 100859x^{28}y^{16} + 165572x^{27}y^{17} + 243067x^{26}y^{18} + 335764x^{25}y^{19} + 420629x^{24}y^{20} + 476476x^{23}y^{21} + 500930x^{22}y^{22} + 481030x^{21}y^{23} + 420596x^{20}y^{24} + 339526x^{19}y^{25} + 247346x^{18}y^{26} + 161304x^{17}y^{27} + 96690x^{16}y^{28} + 53438x^{15}y^{29} + 27929x^{14}y^{30} + 13838x^{13}y^{31} + 5566x^{12}y^{32} + 1608x^{11}y^{33} + 407x^{10}y^{34} + 132x^9y^{35} + 33x^8y^{36}$
$[46, 23, 7]$	17.8398	$x^{46}y^0 + 69x^{39}y^7 + 69x^{38}y^8 + 92x^{37}y^9 + 690x^{36}y^{10} + 1426x^{35}y^{11} + 3634x^{34}y^{12} + 12351x^{33}y^{13} + 29049x^{32}y^{14} + 59248x^{31}y^{15} + 120106x^{30}y^{16} + 212635x^{29}y^{17} + 333753x^{28}y^{18} + 495167x^{27}y^{19} + 670335x^{26}y^{20} + 821928x^{25}y^{21} + 938538x^{24}y^{22} + 982653x^{23}y^{23} + 938469x^{22}y^{24} + 832002x^{21}y^{25} + 674820x^{20}y^{26} + 496248x^{19}y^{27} + 334328x^{18}y^{28} + 203711x^{17}y^{29} + 114701x^{16}y^{30} + 62238x^{15}y^{31} + 30843x^{14}y^{32} + 13271x^{13}y^{33} + 4577x^{12}y^{34} + 1127x^{11}y^{35} + 391x^{10}y^{36} + 138x^9y^{37}$
$[48, 24, 7]$	20.5026	$x^{48}y^0 + 72x^{41}y^7 + 72x^{40}y^8 + 80x^{39}y^9 + 648x^{38}y^{10} + 1296x^{37}y^{11} + 3206x^{36}y^{12} + 11592x^{35}y^{13} + 28704x^{34}y^{14} + 62120x^{33}y^{15} + 135750x^{32}y^{16} + 258816x^{31}y^{17} + 434896x^{30}y^{18} + 689472x^{29}y^{19} + 1001730x^{28}y^{20} + 1321816x^{27}y^{21} + 1626696x^{26}y^{22} + 1846152x^{25}y^{23} + 1916568x^{24}y^{24} + 1846800x^{23}y^{25} + 1641384x^{22}y^{26} + 1335632x^{21}y^{27} + 1002378x^{20}y^{28} + 685560x^{19}y^{29} + 426032x^{18}y^{30} + 247896x^{17}y^{31} + 135321x^{16}y^{32} + 68512x^{15}y^{33} + 31488x^{14}y^{34} + 11616x^{13}y^{35} + 3374x^{12}y^{36} + 1128x^{11}y^{37} + 360x^{10}y^{38} + 48x^9y^{39}$
$[50, 25, 7]$	23.5483	$x^{50}y^0 + 75x^{43}y^7 + 75x^{42}y^8 + 75x^{41}y^9 + 625x^{40}y^{10} + 1200x^{39}y^{11} + 2800x^{38}y^{12} + 10750x^{37}y^{13} + 27675x^{36}y^{14} + 62430x^{35}y^{15} + 146525x^{34}y^{16} + 299725x^{33}y^{17} + 537525x^{32}y^{18} + 911200x^{31}y^{19} + 1415630x^{30}y^{20} + 1997700x^{29}y^{21} + 2635575x^{28}y^{22} + 3218900x^{27}y^{23} + 3609675x^{26}y^{24} + 3764181x^{25}y^{25} + 3631875x^{24}y^{26} + 3226400x^{23}y^{27} + 2658900x^{22}y^{28} + 2016350x^{21}y^{29} + 1397185x^{20}y^{30} + 893250x^{19}y^{31} + 528050x^{18}y^{32} + 292275x^{17}y^{33} + 153175x^{16}y^{34} + 72080x^{15}y^{35} + 28750x^{14}y^{36} + 9600x^{13}y^{37} + 2925x^{12}y^{38} + 1025x^{11}y^{39} + 250x^{10}y^{40}$
$[52, 26, 7]$	27.0115	$x^{52}y^0 + 78x^{45}y^7 + 78x^{44}y^8 + 78x^{43}y^9 + 624x^{42}y^{10} + 1144x^{41}y^{11} + 2444x^{40}y^{12} + 9802x^{39}y^{13} + 26208x^{38}y^{14} + 60970x^{37}y^{15} + 152035x^{36}y^{16} + 332098x^{35}y^{17} + 632853x^{34}y^{18} + 1144156x^{33}y^{19} + 1899794x^{32}y^{20} + 2858362x^{31}y^{21} + 4021654x^{30}y^{22} + 5257746x^{29}y^{23} + 6328127x^{28}y^{24} + 7098910x^{27}y^{25} + 7397703x^{26}y^{26} + 7118176x^{25}y^{27} + 6371560x^{24}y^{28} + 5287022x^{23}y^{29} + 4039620x^{22}y^{30} + 2854150x^{21}y^{31} + 1857180x^{20}y^{32} + 1116102x^{19}y^{33} + 633555x^{18}y^{34} + 336908x^{17}y^{35} + 163098x^{16}y^{36} + 69966x^{15}y^{37} + 24934x^{14}y^{38} + 7984x^{13}y^{39} + 2899x^{12}y^{40} + 780x^{11}y^{41} + 65x^{10}y^{42}$
$[54, 27, 7]$	30.9509	$x^{54}y^0 + 81x^{47}y^7 + 81x^{46}y^8 + 81x^{45}y^9 + 648x^{44}y^{10} + 1134x^{43}y^{11} + 2169x^{42}y^{12} + 8910x^{41}y^{13} + 24354x^{40}y^{14} + 58113x^{39}y^{15} + 153063x^{38}y^{16} + 354105x^{37}y^{17} + 713430x^{36}y^{18} + 1370817x^{35}y^{19} + 2426301x^{34}y^{20} + 3886677x^{33}y^{21} + 5816394x^{32}y^{22} + 8102727x^{31}y^{23} + 10414179x^{30}y^{24} + 12497031x^{29}y^{25} + 13981518x^{28}y^{26} + 14493721x^{27}y^{27} + 14000715x^{26}y^{28} + 12587589x^{25}y^{29} + 10483398x^{24}y^{30} + 8112123x^{23}y^{31} + 5809644x^{22}y^{32} + 3835539x^{21}y^{33} + 2359746x^{20}y^{34} + 1360611x^{19}y^{35} + 734655x^{18}y^{36} + 368847x^{17}y^{37} + 164430x^{16}y^{38} + 62892x^{15}y^{39} + 21816x^{14}y^{40} + 7452x^{13}y^{41} + 2322x^{12}y^{42} + 405x^{11}y^{43} + 9x^9y^{45}$

$[56, 28, 7]$	35.4403	$x^{56}y^0 + 84x^{49}y^7 + 84x^{48}y^8 + 84x^{47}y^9 + 672x^{46}y^{10} + 1176x^{45}y^{11} + 2009x^{44}y^{12} + 8176x^{43}y^{13} + 22554x^{42}y^{14} + 54068x^{41}y^{15} + 149919x^{40}y^{16} + 365932x^{39}y^{17} + 773892x^{38}y^{18} + 1574720x^{37}y^{19} + 2961595x^{36}y^{20} + 5035720x^{35}y^{21} + 8002050x^{34}y^{22} + 11847668x^{33}y^{23} + 16195669x^{32}y^{24} + 20695500x^{31}y^{25} + 24724560x^{30}y^{26} + 27456184x^{29}y^{27} + 28475949x^{28}y^{28} + 27569360x^{27}y^{29} + 24816106x^{26}y^{30} + 20830124x^{25}y^{31} + 16273908x^{24}y^{32} + 11775316x^{23}y^{33} + 7918932x^{22}y^{34} + 4958384x^{21}y^{35} + 2900737x^{20}y^{36} + 1602440x^{19}y^{37} + 822430x^{18}y^{38} + 379736x^{17}y^{39} + 155715x^{16}y^{40} + 56224x^{15}y^{41} + 19448x^{14}y^{42} + 6720x^{13}y^{43} + 1470x^{12}y^{44} + 112x^{11}y^{45} + 28x^{10}y^{46}$
$[58, 29, 7]$	40.5623	$x^{58}y^0 + 87x^{51}y^7 + 87x^{50}y^8 + 87x^{49}y^9 + 696x^{48}y^{10} + 1218x^{47}y^{11} + 1972x^{46}y^{12} + 7714x^{45}y^{13} + 21054x^{44}y^{14} + 49793x^{43}y^{15} + 143144x^{42}y^{16} + 367372x^{41}y^{17} + 812348x^{40}y^{18} + 1741885x^{39}y^{19} + 3470343x^{38}y^{20} + 6242714x^{37}y^{21} + 10503162x^{36}y^{22} + 16495925x^{35}y^{23} + 23922187x^{34}y^{24} + 32437254x^{33}y^{25} + 41204824x^{32}y^{26} + 48770025x^{31}y^{27} + 54027319x^{30}y^{28} + 56035802x^{29}y^{29} + 54200362x^{28}y^{30} + 49004577x^{27}y^{31} + 41402430x^{26}y^{32} + 32563636x^{25}y^{33} + 23871292x^{24}y^{34} + 16295999x^{23}y^{35} + 10352797x^{22}y^{36} + 6177174x^{21}y^{37} + 3469734x^{20}y^{38} + 1814936x^{19}y^{39} + 871450x^{18}y^{40} + 373027x^{17}y^{41} + 141752x^{16}y^{42} + 51417x^{15}y^{43} + 17777x^{14}y^{44} + 4756x^{13}y^{45} + 696x^{12}y^{46} + 58x^{11}y^{47} + 29x^{10}y^{48}$
$[60, 30, 7]$	46.4075	$x^{60}y^0 + 90x^{53}y^7 + 90x^{52}y^8 + 90x^{51}y^9 + 720x^{50}y^{10} + 1260x^{49}y^{11} + 1965x^{48}y^{12} + 7530x^{47}y^{13} + 20130x^{46}y^{14} + 45886x^{45}y^{15} + 134655x^{44}y^{16} + 359610x^{43}y^{17} + 826920x^{42}y^{18} + 1863690x^{41}y^{19} + 3918936x^{40}y^{20} + 7434130x^{39}y^{21} + 13205160x^{38}y^{22} + 21940110x^{37}y^{23} + 33680920x^{36}y^{24} + 48343566x^{35}y^{25} + 65081400x^{34}y^{26} + 81776330x^{33}y^{27} + 96322650x^{32}y^{28} + 106495230x^{31}y^{29} + 110130156x^{30}y^{30} + 106693230x^{29}y^{31} + 96860715x^{28}y^{32} + 82167910x^{27}y^{33} + 65190360x^{26}y^{34} + 48332478x^{25}y^{35} + 33390160x^{24}y^{36} + 21576390x^{23}y^{37} + 13089960x^{22}y^{38} + 7459680x^{21}y^{39} + 3994014x^{20}y^{40} + 1971840x^{19}y^{41} + 876280x^{18}y^{42} + 352770x^{17}y^{43} + 131025x^{16}y^{44} + 46272x^{15}y^{45} + 14370x^{14}y^{46} + 2700x^{13}y^{47} + 325x^{12}y^{48} + 120x^{11}y^{49}$
$[70, 35, 7]$	90.4576	$x^{70}y^0 + 105x^{63}y^7 + 105x^{62}y^8 + 105x^{61}y^9 + 840x^{60}y^{10} + 1470x^{59}y^{11} + 2205x^{58}y^{12} + 8085x^{57}y^{13} + 21210x^{56}y^{14} + 40257x^{55}y^{15} + 106575x^{54}y^{16} + 296205x^{53}y^{17} + 704375x^{52}y^{18} + 1812370x^{51}y^{19} + 4676007x^{50}y^{20} + 10926585x^{49}y^{21} + 24356430x^{48}y^{22} + 51736405x^{47}y^{23} + 101793720x^{46}y^{24} + 187639137x^{45}y^{25} + 326311860x^{44}y^{26} + 531964020x^{43}y^{27} + 815554010x^{42}y^{28} + 1180672850x^{41}y^{29} + 1611463196x^{40}y^{30} + 2075929870x^{39}y^{31} + 2529360995x^{38}y^{32} + 2912761670x^{37}y^{33} + 3170836970x^{36}y^{34} + 3265237884x^{35}y^{35} + 3177663510x^{34}y^{36} + 2921506770x^{33}y^{37} + 2537893540x^{32}y^{38} + 2081061255x^{31}y^{39} + 1610335573x^{30}y^{40} + 1176086975x^{29}y^{41} + 810246860x^{28}y^{42} + 527159150x^{27}y^{43} + 324453465x^{26}y^{44} + 188737241x^{25}y^{45} + 103504170x^{24}y^{46} + 53141305x^{23}y^{47} + 25302725x^{22}y^{48} + 11184885x^{21}y^{49} + 4618327x^{20}y^{50} + 1774290x^{19}y^{51} + 617715x^{18}y^{52} + 179445x^{17}y^{53} + 42350x^{16}y^{54} + 10675x^{15}y^{55} + 2450x^{14}y^{56} + 175x^{13}y^{57}$

$[80, 40, 7]$	174.9980	$x^{80}y^0+120x^{73}y^7+120x^{72}y^8+120x^{71}y^9+960x^{70}y^{10}+1680x^{69}y^{11}+2520x^{68}y^{12}+9240x^{67}y^{13}+25140x^{66}y^{14}+46608x^{65}y^{15}+114900x^{64}y^{16}+311920x^{63}y^{17}+676600x^{62}y^{18}+1604280x^{61}y^{19}+4216918x^{60}y^{20}+10298200x^{59}y^{21}+25177080x^{58}y^{22}+62234280x^{57}y^{23}+145540985x^{56}y^{24}+323515048x^{55}y^{25}+689445920x^{54}y^{26}+1387438640x^{53}y^{27}+2634314620x^{52}y^{28}+4743816360x^{51}y^{29}+8085095208x^{50}y^{30}+13043101160x^{49}y^{31}+19972142375x^{48}y^{32}+29035879480x^{47}y^{33}+40091439060x^{46}y^{34}+52647124880x^{45}y^{35}+65773046420x^{44}y^{36}+78185347920x^{43}y^{37}+88483096640x^{42}y^{38}+95336248680x^{41}y^{39}+97776703370x^{40}y^{40}+95457848360x^{39}y^{41}+88685518640x^{38}y^{42}+78378398480x^{37}y^{43}+65886901880x^{36}y^{44}+52666129096x^{35}y^{45}+40021487700x^{34}y^{46}+28916032640x^{33}y^{47}+19864604670x^{32}y^{48}+12978425280x^{31}y^{49}+8067080624x^{30}y^{50}+4766237560x^{29}y^{51}+2670902950x^{28}y^{52}+1415044760x^{27}y^{53}+706054120x^{26}y^{54}+331412536x^{25}y^{55}+146426405x^{24}y^{56}+60574200x^{23}y^{57}+23143440x^{22}y^{58}+7986800x^{21}y^{59}+2459812x^{20}y^{60}+706360x^{19}y^{61}+190000x^{18}y^{62}+38600x^{17}y^{63}+4710x^{16}y^{64}+600x^{15}y^{65}+100x^{14}y^{66}$
$[90, 45, 7]$	337.0054	$x^{90}y^0+135x^{83}y^7+135x^{82}y^8+135x^{81}y^9+1080x^{80}y^{10}+1890x^{79}y^{11}+2835x^{78}y^{12}+10395x^{77}y^{13}+29295x^{76}y^{14}+54450x^{75}y^{15}+132075x^{74}y^{16}+367110x^{73}y^{17}+786750x^{72}y^{18}+1769175x^{71}y^{19}+4515039x^{70}y^{20}+10521450x^{69}y^{21}+24412050x^{68}y^{22}+60567255x^{67}y^{23}+147589290x^{66}y^{24}+352830114x^{65}y^{25}+843604380x^{64}y^{26}+1955776430x^{63}y^{27}+4343399235x^{62}y^{28}+9277303140x^{61}y^{29}+18930475761x^{60}y^{30}+36747082485x^{59}y^{31}+67993216515x^{58}y^{32}+119887096605x^{57}y^{33}+201320091450x^{56}y^{34}+322389384966x^{55}y^{35}+492692869895x^{54}y^{36}+718778623905x^{53}y^{37}+1001853697680x^{52}y^{38}+1334966751450x^{51}y^{39}+1701039353058x^{50}y^{40}+2073594668265x^{49}y^{41}+2418937634880x^{48}y^{42}+2700504124515x^{47}y^{43}+2885560564545x^{46}y^{44}+2951084430699x^{45}y^{45}+2888247114945x^{44}y^{46}+2704809830220x^{43}y^{47}+2423352318165x^{42}y^{48}+2076713276040x^{41}y^{49}+1702003921938x^{40}y^{50}+1333890973755x^{39}y^{51}+999562979925x^{38}y^{52}+716211293220x^{37}y^{53}+490709350090x^{36}y^{54}+321465679479x^{35}y^{55}+201321509490x^{34}y^{56}+120437748240x^{33}y^{57}+68725801260x^{32}y^{58}+37339931700x^{31}y^{59}+19278641769x^{30}y^{60}+9444823110x^{29}y^{61}+4384927935x^{28}y^{62}+1921922525x^{27}y^{63}+789515100x^{26}y^{64}+301395429x^{25}y^{65}+106361910x^{24}y^{66}+34860600x^{23}y^{67}+10586565x^{22}y^{68}+2825505x^{21}y^{69}+616500x^{20}y^{70}+116865x^{19}y^{71}+22875x^{18}y^{72}+3150x^{17}y^{73}+9x^{15}y^{75}$

$[100, 50, 7]$	647.1535	$x^{100}y^0 + 150x^{93}y^7 + 150x^{92}y^8 + 150x^{91}y^9 + 1200x^{90}y^{10} + 2100x^{89}y^{11} + 3150x^{88}y^{12} + 11550x^{87}y^{13} + 33675x^{86}y^{14} + 62750x^{85}y^{15} + 150125x^{84}y^{16} + 428150x^{83}y^{17} + 924325x^{82}y^{18} + 2056200x^{81}y^{19} + 5295525x^{80}y^{20} + 12311650x^{79}y^{21} + 27631600x^{78}y^{22} + 66560100x^{77}y^{23} + 157868400x^{76}y^{24} + 366405410x^{75}y^{25} + 876044050x^{74}y^{26} + 2101084000x^{73}y^{27} + 4955643925x^{72}y^{28} + 11589560800x^{71}y^{29} + 26553866070x^{70}y^{30} + 58825947950x^{69}y^{31} + 125839459775x^{68}y^{32} + 259067997700x^{67}y^{33} + 511159257125x^{66}y^{34} + 966226061600x^{65}y^{35} + 1749628414125x^{64}y^{36} + 3033227631150x^{63}y^{37} + 5036548501800x^{62}y^{38} + 8014837444900x^{61}y^{39} + 12227091713590x^{60}y^{40} + 17891316860450x^{59}y^{41} + 25124386374200x^{58}y^{42} + 33872075838400x^{57}y^{43} + 43857381274025x^{56}y^{44} + 54556761310240x^{55}y^{45} + 65215382744900x^{54}y^{46} + 74923615368950x^{53}y^{47} + 82738524328325x^{52}y^{48} + 87826925593500x^{51}y^{49} + 89613078314961x^{50}y^{50} + 87885619091400x^{49}y^{51} + 82836069218775x^{48}y^{52} + 75028071071550x^{47}y^{53} + 65294860489200x^{46}y^{54} + 54591933454260x^{45}y^{55} + 43845936892300x^{44}y^{56} + 33826177497150x^{43}y^{57} + 25065269567450x^{42}y^{58} + 17839020668400x^{41}y^{59} + 12193340462455x^{40}y^{60} + 8003013590600x^{39}y^{61} + 5042187706650x^{38}y^{62} + 3047535867250x^{37}y^{63} + 1765301107600x^{36}y^{64} + 978850588740x^{35}y^{65} + 518907709675x^{34}y^{66} + 262643280400x^{33}y^{67} + 126724497175x^{32}y^{68} + 58132897650x^{31}y^{69} + 25251596600x^{30}y^{70} + 10340874350x^{29}y^{71} + 3979031000x^{28}y^{72} + 1435935400x^{27}y^{73} + 483586700x^{26}y^{74} + 149345212x^{25}y^{75} + 41284925x^{24}y^{76} + 10236250x^{23}y^{77} + 2359425x^{22}y^{78} + 479500x^{21}y^{79} + 65310x^{20}y^{80} + 5350x^{19}y^{81} + 1050x^{18}y^{82}$

$[104, 52, 7]$	839.6969	$x^{104}y^0 + 156x^{97}y^7 + 156x^{96}y^8 + 156x^{95}y^9 + 1248x^{94}y^{10} + 2184x^{93}y^{11} + 3276x^{92}y^{12} + 12012x^{91}y^{13} + 35490x^{90}y^{14} + 66196x^{89}y^{15} + 157534x^{88}y^{16} + 453700x^{87}y^{17} + 982358x^{86}y^{18} + 2178696x^{85}y^{19} + 5641948x^{84}y^{20} + 13180128x^{83}y^{21} + 29505528x^{82}y^{22} + 70946512x^{81}y^{23} + 167740456x^{80}y^{24} + 384824440x^{79}y^{25} + 909149306x^{78}y^{26} + 2165260864x^{77}y^{27} + 5094924380x^{76}y^{28} + 12013920672x^{75}y^{29} + 28105116182x^{74}y^{30} + 64203359532x^{73}y^{31} + 142890140211x^{72}y^{32} + 308335063244x^{71}y^{33} + 640927733238x^{70}y^{34} + 1281152637752x^{69}y^{35} + 2460297285329x^{68}y^{36} + 4532562929232x^{67}y^{37} + 8009551530916x^{66}y^{38} + 13580919665824x^{65}y^{39} + 22097933505310x^{64}y^{40} + 34517923201528x^{63}y^{41} + 51789616320962x^{62}y^{42} + 74664268959280x^{61}y^{43} + 103473771512202x^{60}y^{44} + 137903922962728x^{59}y^{45} + 176800428787294x^{58}y^{46} + 218103154120124x^{57}y^{47} + 258946765010167x^{56}y^{48} + 295929126088076x^{55}y^{49} + 325558810950698x^{54}y^{50} + 344790805560968x^{53}y^{51} + 351526636108731x^{52}y^{52} + 344995130742496x^{51}y^{53} + 325901756578320x^{50}y^{54} + 296300213463904x^{49}y^{55} + 259238303725388x^{48}y^{56} + 218244833619384x^{47}y^{57} + 176776099223438x^{46}y^{58} + 137754385508480x^{45}y^{59} + 103268346077624x^{44}y^{60} + 74470373462368x^{43}y^{61} + 51656248163546x^{42}y^{62} + 34460942820980x^{41}y^{63} + 22104181713276x^{40}y^{64} + 13625664630452x^{39}y^{65} + 8066035160690x^{38}y^{66} + 4580957673448x^{37}y^{67} + 2493294498695x^{36}y^{68} + 1298995618960x^{35}y^{69} + 646925335316x^{34}y^{70} + 307381326356x^{33}y^{71} + 138936179954x^{32}y^{72} + 59522244028x^{31}y^{73} + 24091988310x^{30}y^{74} + 9190224264x^{29}y^{75} + 3292533686x^{28}y^{76} + 1098371820x^{27}y^{77} + 335658832x^{26}y^{78} + 92920464x^{25}y^{79} + 23620571x^{24}y^{80} + 5514288x^{23}y^{81} + 1066520x^{22}y^{82} + 146640x^{21}y^{83} + 18161x^{20}y^{84} + 2912x^{19}y^{85}$

$[108, 54, 7]$	1089.2798	$x^{108}y^0 + 162x^{101}y^7 + 162x^{100}y^8 + 162x^{99}y^9 + 1296x^{98}y^{10} + 2268x^{97}y^{11} + 3402x^{96}y^{12} + 12474x^{95}y^{13} + 37341x^{94}y^{14} + 69714x^{93}y^{15} + 165051x^{92}y^{16} + 479898x^{91}y^{17} + 1042011x^{90}y^{18} + 2304288x^{89}y^{19} + 5998914x^{88}y^{20} + 14086602x^{87}y^{21} + 31512888x^{86}y^{22} + 75861252x^{85}y^{23} + 179627031x^{84}y^{24} + 409704696x^{83}y^{25} + 959694750x^{82}y^{26} + 2268079296x^{81}y^{27} + 5292175878x^{80}y^{28} + 12441425688x^{79}y^{29} + 29310629301x^{78}y^{30} + 68064685668x^{77}y^{31} + 155516575671x^{76}y^{32} + 347745799458x^{75}y^{33} + 754311246093x^{74}y^{34} + 1581912909162x^{73}y^{35} + 3200768643768x^{72}y^{36} + 6231770810478x^{71}y^{37} + 11663364853485x^{70}y^{38} + 20981051593614x^{69}y^{39} + 36265718366775x^{68}y^{40} + 60239528998200x^{67}y^{41} + 96197220187245x^{66}y^{42} + 147733245224676x^{65}y^{43} + 218268134914086x^{64}y^{44} + 310376465061216x^{63}y^{45} + 424939313329155x^{62}y^{46} + 560327252145228x^{61}y^{47} + 711810364744971x^{60}y^{48} + 871350599944746x^{59}y^{49} + 1028021454078903x^{58}y^{50} + 1169098645092750x^{57}y^{51} + 1281657349680624x^{56}y^{52} + 1354490236720926x^{55}y^{53} + 1379942168659513x^{54}y^{54} + 1355209503879726x^{53}y^{55} + 1282862084182857x^{52}y^{56} + 1170425955478464x^{51}y^{57} + 1029092865111333x^{50}y^{58} + 871904327279400x^{49}y^{59} + 711787585749210x^{48}y^{60} + 559843427769264x^{47}y^{61} + 424219384859895x^{46}y^{62} + 309669586403796x^{45}y^{63} + 217749167631396x^{44}y^{64} + 147471932148894x^{43}y^{65} + 96174854736399x^{42}y^{66} + 60373515159270x^{41}y^{67} + 36459823302000x^{40}y^{68} + 21165623646930x^{39}y^{69} + 11800179239607x^{38}y^{70} + 6311637624960x^{37}y^{71} + 3235059827379x^{36}y^{72} + 1586473488558x^{35}y^{73} + 742776181623x^{34}y^{74} + 331081645464x^{33}y^{75} + 140073206256x^{32}y^{76} + 56108674782x^{31}y^{77} + 21220600500x^{30}y^{78} + 7537137642x^{29}y^{79} + 2489995998x^{28}y^{80} + 756538940x^{27}y^{81} + 211011858x^{26}y^{82} + 54434754x^{25}y^{83} + 12572262x^{24}y^{84} + 2362392x^{23}y^{85} + 357291x^{22}y^{86} + 55134x^{21}y^{87} + 6804x^{20}y^{88} + 9x^{18}y^{90}$

B.4 Weight enumerator for constraint length $m = 5$

Secrecy gain: Constraint length 5		
$[n, k, d]$	Secrecy gain	$W_{\mathscr{C}}(x, y)$
$[12, 6, 4]$	1.6569	$x^{12}y^0 + 6x^8y^4 + 24x^7y^5 + 16x^6y^6 + 9x^4y^8 + 8x^3y^9$
$[14, 7, 4]$	1.8283	$x^{14}y^0 + 7x^{10}y^4 + 21x^9y^5 + 21x^8y^6 + 29x^7y^7 + 28x^6y^8 + 7x^5y^9 + 7x^4y^{10} + 7x^3y^{11}$
$[16, 8, 5]$	2.1410	$x^{16}y^0 + 24x^{11}y^5 + 44x^{10}y^6 + 40x^9y^7 + 45x^8y^8 + 40x^7y^9 + 28x^6y^{10} + 24x^5y^{11} + 10x^4y^{12}$
$[18, 9, 6]$	2.4854	$x^{18}y^0 + 102x^{12}y^6 + 153x^{10}y^8 + 153x^8y^{10} + 102x^6y^{12} + 1x^0y^{18}$
$[20, 10, 6]$	2.8682	$x^{20}y^0 + 40x^{14}y^6 + 160x^{13}y^7 + 130x^{12}y^8 + 176x^{10}y^{10} + 320x^9y^{11} + 120x^8y^{12} + 40x^6y^{14} + 32x^5y^{15} + 5x^4y^{16}$
$[22, 11, 7]$	3.3347	$x^{22}y^0 + 176x^{15}y^7 + 330x^{14}y^8 + 672x^{11}y^{11} + 616x^{10}y^{12} + 176x^7y^{15} + 77x^6y^{16}$
$[24, 12, 6]$	3.7498	$x^{24}y^0 + 20x^{18}y^6 + 120x^{17}y^7 + 234x^{16}y^8 + 304x^{15}y^9 + 372x^{14}y^{10} + 552x^{13}y^{11} + 808x^{12}y^{12} + 672x^{11}y^{13} + 348x^{10}y^{14} + 328x^9y^{15} + 237x^8y^{16} + 48x^7y^{17} + 28x^6y^{18} + 24x^5y^{19}$
$[26, 13, 7]$	4.3555	$x^{26}y^0 + 117x^{19}y^7 + 273x^{18}y^8 + 338x^{17}y^9 + 598x^{16}y^{10} + 923x^{15}y^{11} + 1105x^{14}y^{12} + 1340x^{13}y^{13} + 1300x^{12}y^{14} + 923x^{11}y^{15} + 598x^{10}y^{16} + 338x^9y^{17} + 182x^8y^{18} + 117x^7y^{19} + 39x^6y^{20}$
$[28, 14, 7]$	5.0441	$x^{28}y^0 + 56x^{21}y^7 + 301x^{20}y^8 + 504x^{19}y^9 + 672x^{18}y^{10} + 1176x^{17}y^{11} + 1736x^{16}y^{12} + 2520x^{15}y^{13} + 2752x^{14}y^{14} + 1960x^{13}y^{15} + 1771x^{12}y^{16} + 1512x^{11}y^{17} + 672x^{10}y^{18} + 392x^9y^{19} + 280x^8y^{20} + 72x^7y^{21} + 7x^4y^{24}$
$[30, 15, 8]$	5.8431	$x^{30}y^0 + 450x^{22}y^8 + 1848x^{20}y^{10} + 5040x^{18}y^{12} + 9045x^{16}y^{14} + 9045x^{14}y^{16} + 5040x^{12}y^{18} + 1848x^{10}y^{20} + 450x^8y^{22} + 1x^0y^{30}$
$[32, 16, 7]$	6.7569	$x^{32}y^0 + 16x^{25}y^7 + 184x^{24}y^8 + 592x^{23}y^9 + 1008x^{22}y^{10} + 1808x^{21}y^{11} + 3276x^{20}y^{12} + 5168x^{19}y^{13} + 7512x^{18}y^{14} + 8880x^{17}y^{15} + 8989x^{16}y^{16} + 8656x^{15}y^{17} + 7016x^{14}y^{18} + 5168x^{13}y^{19} + 3708x^{12}y^{20} + 1936x^{11}y^{21} + 936x^{10}y^{22} + 512x^9y^{23} + 98x^8y^{24} + 32x^7y^{25} + 40x^6y^{26}$
$[34, 17, 8]$	7.8446	$x^{34}y^0 + 170x^{26}y^8 + 459x^{25}y^9 + 1122x^{24}y^{10} + 2261x^{23}y^{11} + 3672x^{22}y^{12} + 7089x^{21}y^{13} + 10948x^{20}y^{14} + 13872x^{19}y^{15} + 17357x^{18}y^{16} + 17834x^{17}y^{17} + 16184x^{16}y^{18} + 14450x^{15}y^{19} + 10608x^{14}y^{20} + 7038x^{13}y^{21} + 4284x^{12}y^{22} + 2040x^{11}y^{23} + 1088x^{10}y^{24} + 459x^9y^{25} + 102x^8y^{26} + 17x^7y^{27} + 17x^5y^{29}$
$[36, 18, 8]$	9.08210	$x^{36}y^0 + 117x^{28}y^8 + 452x^{27}y^9 + 990x^{26}y^{10} + 2232x^{25}y^{11} + 4740x^{24}y^{12} + 9108x^{23}y^{13} + 14490x^{22}y^{14} + 19872x^{21}y^{15} + 27963x^{20}y^{16} + 35208x^{19}y^{17} + 34540x^{18}y^{18} + 30672x^{17}y^{19} + 27792x^{16}y^{20} + 22344x^{15}y^{21} + 14580x^{14}y^{22} + 8352x^{13}y^{23} + 4815x^{12}y^{24} + 2484x^{11}y^{25} + 918x^{10}y^{26} + 312x^9y^{27} + 108x^8y^{28} + 36x^7y^{29} + 18x^6y^{30}$
$[38, 19, 8]$	10.5528	$x^{38}y^0 + 76x^{30}y^8 + 380x^{29}y^9 + 912x^{28}y^{10} + 2337x^{27}y^{11} + 5244x^{26}y^{12} + 10165x^{25}y^{13} + 18525x^{24}y^{14} + 29146x^{23}y^{15} + 41933x^{22}y^{16} + 56012x^{21}y^{17} + 64372x^{20}y^{18} + 66710x^{19}y^{19} + 64372x^{18}y^{20} + 54454x^{17}y^{21} + 41990x^{16}y^{22} + 30248x^{15}y^{23} + 18430x^{14}y^{24} + 10032x^{13}y^{25} + 5396x^{12}y^{26} + 2337x^{11}y^{27} + 760x^{10}y^{28} + 285x^9y^{29} + 133x^8y^{30} + 38x^7y^{31}$
$[40, 20, 8]$	12.2480	$x^{40}y^0 + 100x^{32}y^8 + 1760x^{30}y^{10} + 10710x^{28}y^{12} + 43680x^{26}y^{14} + 120715x^{24}y^{16} + 215680x^{22}y^{18} + 263284x^{20}y^{20} + 215680x^{18}y^{22} + 120715x^{16}y^{24} + 43680x^{14}y^{26} + 10710x^{12}y^{28} + 1760x^{10}y^{30} + 100x^8y^{32} + 1x^0y^{40}$

$[42, 21, 8]$	14.13510	$x^{42}y^0+84x^{34}y^8+231x^{33}y^9+525x^{32}y^{10}+2142x^{31}y^{11}+5446x^{30}y^{12}+11865x^{29}y^{13}+25620x^{28}y^{14}+47313x^{27}y^{15}+78876x^{26}y^{16}+121758x^{25}y^{17}+167853x^{24}y^{18}+211701x^{23}y^{19}+246540x^{22}y^{20}+258290x^{21}y^{21}+244608x^{20}y^{22}+212730x^{19}y^{23}+168112x^{18}y^{24}+120435x^{17}y^{25}+79527x^{16}y^{26}+47656x^{15}y^{27}+25518x^{14}y^{28}+12453x^{13}y^{29}+5068x^{12}y^{30}+1701x^{11}y^{31}+735x^{10}y^{32}+280x^9y^{33}+63x^8y^{34}+21x^7y^{35}$
$[44, 22, 8]$	16.5191	$x^{44}y^0+44x^{36}y^8+198x^{35}y^9+616x^{34}y^{10}+1628x^{33}y^{11}+4708x^{32}y^{12}+13156x^{31}y^{13}+27973x^{30}y^{14}+53812x^{29}y^{15}+100089x^{28}y^{16}+163504x^{27}y^{17}+242407x^{26}y^{18}+336820x^{25}y^{19}+422653x^{24}y^{20}+480150x^{23}y^{21}+502338x^{22}y^{22}+478456x^{21}y^{23}+416570x^{20}y^{24}+336710x^{19}y^{25}+248006x^{18}y^{26}+163636x^{17}y^{27}+99022x^{16}y^{28}+54384x^{15}y^{29}+26697x^{14}y^{30}+12980x^{13}y^{31}+5456x^{12}y^{32}+1476x^{11}y^{33}+539x^{10}y^{34}+220x^9y^{35}+33x^8y^{36}+22x^7y^{37}$
$[46, 23, 8]$	19.2623	$x^{46}y^0+46x^{38}y^8+115x^{37}y^9+414x^{36}y^{10}+1817x^{35}y^{11}+4347x^{34}y^{12}+11684x^{33}y^{13}+30015x^{32}y^{14}+60743x^{31}y^{15}+116679x^{30}y^{16}+210450x^{29}y^{17}+335225x^{28}y^{18}+493074x^{27}y^{19}+671692x^{26}y^{20}+828069x^{25}y^{21}+937388x^{24}y^{22}+982446x^{23}y^{23}+942080x^{22}y^{24}+827103x^{21}y^{25}+669760x^{20}y^{26}+495949x^{19}y^{27}+333523x^{18}y^{28}+207046x^{17}y^{29}+119255x^{16}y^{30}+62491x^{15}y^{31}+29210x^{14}y^{32}+11500x^{13}y^{33}+4025x^{12}y^{34}+1656x^{11}y^{35}+598x^{10}y^{36}+161x^9y^{37}+46x^8y^{38}$
$[48, 24, 8]$	22.2955	$x^{48}y^0+63x^{40}y^8+72x^{39}y^9+456x^{38}y^{10}+840x^{37}y^{11}+5420x^{36}y^{12}+9384x^{35}y^{13}+30420x^{34}y^{14}+64552x^{33}y^{15}+133707x^{32}y^{16}+256368x^{31}y^{17}+432556x^{30}y^{18}+695904x^{29}y^{19}+982620x^{28}y^{20}+1335584x^{27}y^{21}+1637496x^{26}y^{22}+1830384x^{25}y^{23}+1944177x^{24}y^{24}+1834920x^{23}y^{25}+1634688x^{22}y^{26}+1333000x^{21}y^{27}+985482x^{20}y^{28}+694824x^{19}y^{29}+430628x^{18}y^{30}+259656x^{17}y^{31}+135180x^{16}y^{32}+62240x^{15}y^{33}+29580x^{14}y^{34}+9936x^{13}y^{35}+5576x^{12}y^{36}+912x^{11}y^{37}+528x^{10}y^{38}+32x^9y^{39}+24x^8y^{40}+6x^4y^{44}$
$[50, 25, 8]$	25.8932	$x^{50}y^0+50x^{42}y^8+125x^{41}y^9+300x^{40}y^{10}+1000x^{39}y^{11}+3375x^{38}y^{12}+10675x^{37}y^{13}+27075x^{36}y^{14}+67050x^{35}y^{15}+150525x^{34}y^{16}+291825x^{33}y^{17}+536275x^{32}y^{18}+912700x^{31}y^{19}+1398035x^{30}y^{20}+1998675x^{29}y^{21}+2653175x^{28}y^{22}+3218150x^{27}y^{23}+3620675x^{26}y^{24}+3777691x^{25}y^{25}+3622975x^{24}y^{26}+3216400x^{23}y^{27}+2640925x^{22}y^{28}+1999025x^{21}y^{29}+1408025x^{20}y^{30}+913750x^{19}y^{31}+537350x^{18}y^{32}+291475x^{17}y^{33}+146025x^{16}y^{34}+66700x^{15}y^{35}+29025x^{14}y^{36}+10825x^{13}y^{37}+2925x^{12}y^{38}+1050x^{11}y^{39}+455x^{10}y^{40}+100x^9y^{41}+25x^8y^{42}$
$[52, 26, 8]$	30.0883	$x^{52}y^0+52x^{44}y^8+104x^{43}y^9+286x^{42}y^{10}+910x^{41}y^{11}+2496x^{40}y^{12}+9620x^{39}y^{13}+26247x^{38}y^{14}+64298x^{37}y^{15}+157469x^{36}y^{16}+331656x^{35}y^{17}+631358x^{34}y^{18}+1138592x^{33}y^{19}+1883362x^{32}y^{20}+2852980x^{31}y^{21}+4021485x^{30}y^{22}+5261646x^{29}y^{23}+6356649x^{28}y^{24}+7121192x^{27}y^{25}+7404580x^{26}y^{26}+7111000x^{25}y^{27}+6336044x^{24}y^{28}+5258812x^{23}y^{29}+4034641x^{22}y^{30}+2864966x^{21}y^{31}+1881698x^{20}y^{32}+1130584x^{19}y^{33}+633334x^{18}y^{34}+331448x^{17}y^{35}+155922x^{16}y^{36}+67132x^{15}y^{37}+25051x^{14}y^{38}+8322x^{13}y^{39}+3523x^{12}y^{40}+1040x^{11}y^{41}+234x^{10}y^{42}+130x^9y^{43}$

$[54, 27, 8]$	34.9928	$x^{54}y^0 + 54x^{46}y^8 + 84x^{45}y^9 + 270x^{44}y^{10} + 810x^{43}y^{11} + 2142x^{42}y^{12} + 7722x^{41}y^{13} + 24354x^{40}y^{14} + 64143x^{39}y^{15} + 153981x^{38}y^{16} + 354213x^{37}y^{17} + 731856x^{36}y^{18} + 1363662x^{35}y^{19} + 2393010x^{34}y^{20} + 3883842x^{33}y^{21} + 5795253x^{32}y^{22} + 8082045x^{31}y^{23} + 10469781x^{30}y^{24} + 12543039x^{29}y^{25} + 13990482x^{28}y^{26} + 14510548x^{27}y^{27} + 13973472x^{26}y^{28} + 12539556x^{25}y^{29} + 10467531x^{24}y^{30} + 8090361x^{23}y^{31} + 5801922x^{22}y^{32} + 3871107x^{21}y^{33} + 2393604x^{20}y^{34} + 1374678x^{19}y^{35} + 731106x^{18}y^{36} + 350298x^{17}y^{37} + 152739x^{16}y^{38} + 63387x^{15}y^{39} + 24597x^{14}y^{40} + 8613x^{13}y^{41} + 2412x^{12}y^{42} + 702x^{11}y^{43} + 270x^{10}y^{44} + 54x^9y^{45} + 27x^8y^{46}$
$[56, 28, 8]$	40.6476	$x^{56}y^0 + 56x^{48}y^8 + 84x^{47}y^9 + 238x^{46}y^{10} + 644x^{45}y^{11} + 2205x^{44}y^{12} + 6496x^{43}y^{13} + 19488x^{42}y^{14} + 61600x^{41}y^{15} + 155589x^{40}y^{16} + 360220x^{39}y^{17} + 798462x^{38}y^{18} + 1591128x^{37}y^{19} + 2916501x^{36}y^{20} + 5020544x^{35}y^{21} + 7992796x^{34}y^{22} + 11775036x^{33}y^{23} + 16216543x^{32}y^{24} + 20783532x^{31}y^{25} + 24734206x^{30}y^{26} + 27512408x^{29}y^{27} + 28535575x^{28}y^{28} + 27513976x^{27}y^{29} + 24764936x^{26}y^{30} + 20762308x^{25}y^{31} + 16188186x^{24}y^{32} + 11785732x^{23}y^{33} + 7986930x^{22}y^{34} + 5035384x^{21}y^{35} + 2943535x^{20}y^{36} + 1575504x^{19}y^{37} + 782684x^{18}y^{38} + 363972x^{17}y^{39} + 156653x^{16}y^{40} + 62048x^{15}y^{41} + 20708x^{14}y^{42} + 6356x^{13}y^{43} + 2184x^{12}y^{44} + 728x^{11}y^{45} + 224x^{10}y^{46} + 28x^9y^{47} + 28x^8y^{48}$
$[58, 29, 8]$	47.2069	$x^{58}y^0 + 58x^{50}y^8 + 87x^{49}y^9 + 232x^{48}y^{10} + 522x^{47}y^{11} + 1711x^{46}y^{12} + 6409x^{45}y^{13} + 17110x^{44}y^{14} + 52722x^{43}y^{15} + 150481x^{42}y^{16} + 366618x^{41}y^{17} + 833837x^{40}y^{18} + 1773321x^{39}y^{19} + 3452798x^{38}y^{20} + 6219514x^{37}y^{21} + 10486951x^{36}y^{22} + 16407475x^{35}y^{23} + 23838928x^{34}y^{24} + 32477970x^{33}y^{25} + 41285125x^{32}y^{26} + 48871989x^{31}y^{27} + 54185920x^{30}y^{28} + 56078896x^{29}y^{29} + 54110723x^{28}y^{30} + 48885735x^{27}y^{31} + 41229387x^{26}y^{32} + 32458540x^{25}y^{33} + 23919519x^{24}y^{34} + 16414435x^{23}y^{35} + 10484138x^{22}y^{36} + 6227054x^{21}y^{37} + 3424349x^{20}y^{38} + 1766593x^{19}y^{39} + 847438x^{18}y^{40} + 367575x^{17}y^{41} + 146247x^{16}y^{42} + 52693x^{15}y^{43} + 18473x^{14}y^{44} + 6815x^{13}y^{45} + 1827x^{12}y^{46} + 435x^{11}y^{47} + 203x^{10}y^{48} + 58x^9y^{49}$
$[60, 30, 8]$	54.7209	$x^{60}y^0 + 60x^{52}y^8 + 90x^{51}y^9 + 255x^{50}y^{10} + 450x^{49}y^{11} + 1755x^{48}y^{12} + 3990x^{47}y^{13} + 18735x^{46}y^{14} + 42380x^{45}y^{15} + 142650x^{44}y^{16} + 350430x^{43}y^{17} + 866650x^{42}y^{18} + 1896330x^{41}y^{19} + 3918540x^{40}y^{20} + 7499820x^{39}y^{21} + 13096380x^{38}y^{22} + 21871770x^{37}y^{23} + 33534340x^{36}y^{24} + 48231000x^{35}y^{25} + 65168010x^{34}y^{26} + 81859640x^{33}y^{27} + 96709770x^{32}y^{28} + 106572120x^{31}y^{29} + 110223746x^{30}y^{30} + 106643460x^{29}y^{31} + 96434145x^{28}y^{32} + 82061760x^{27}y^{33} + 64954560x^{26}y^{34} + 48407496x^{25}y^{35} + 33626460x^{24}y^{36} + 21680820x^{23}y^{37} + 13253580x^{22}y^{38} + 7412800x^{21}y^{39} + 3934680x^{20}y^{40} + 1912830x^{19}y^{41} + 838975x^{18}y^{42} + 361110x^{17}y^{43} + 131955x^{16}y^{44} + 56370x^{15}y^{45} + 14535x^{14}y^{46} + 5520x^{13}y^{47} + 1100x^{12}y^{48} + 690x^{11}y^{49} + 30x^{10}y^{50} + 30x^9y^{51} + 6x^5y^{55}$

$[70, 35, 8]$	114.1804	$x^{70}y^0+70x^{62}y^8+105x^{61}y^9+280x^{60}y^{10}+525x^{59}y^{11}+1435x^{58}y^{12}+3290x^{57}y^{13}+8925x^{56}y^{14}+26607x^{55}y^{15}+73955x^{54}y^{16}+226870x^{53}y^{17}+668080x^{52}y^{18}+1820175x^{51}y^{19}+4632222x^{50}y^{20}+11158315x^{49}y^{21}+24992450x^{48}y^{22}+52064880x^{47}y^{23}+102326665x^{46}y^{24}+188418902x^{45}y^{25}+325217445x^{44}y^{26}+530038495x^{43}y^{27}+813965960x^{42}y^{28}+1177713320x^{41}y^{29}+1610037086x^{40}y^{30}+2078382530x^{39}y^{31}+2532959835x^{38}y^{32}+2917416880x^{37}y^{33}+3175507300x^{36}y^{34}+3265764994x^{35}y^{35}+3174470320x^{34}y^{36}+2916924850x^{33}y^{37}+2532618760x^{32}y^{38}+2077798100x^{31}y^{39}+1610571228x^{30}y^{40}+1178275385x^{29}y^{41}+813760990x^{28}y^{42}+530508055x^{27}y^{43}+325603215x^{26}y^{44}+188067446x^{25}y^{45}+102082365x^{24}y^{46}+51935975x^{23}y^{47}+24921785x^{22}y^{48}+11228770x^{21}y^{49}+4696132x^{20}y^{50}+1833335x^{19}y^{51}+662970x^{18}y^{52}+236075x^{17}y^{53}+82110x^{16}y^{54}+22820x^{15}y^{55}+7005x^{14}y^{56}+2240x^{13}y^{57}+525x^{12}y^{58}+245x^{11}y^{59}+70x^{10}y^{60}$
$[80, 40, 8]$	236.1906	$x^{80}y^0+80x^{72}y^8+120x^{71}y^9+320x^{70}y^{10}+600x^{69}y^{11}+1640x^{68}y^{12}+3600x^{67}y^{13}+9020x^{66}y^{14}+21120x^{65}y^{15}+56360x^{64}y^{16}+143400x^{63}y^{17}+397060x^{62}y^{18}+1166240x^{61}y^{19}+3289348x^{60}y^{20}+9097360x^{59}y^{21}+24323420x^{58}y^{22}+61552200x^{57}y^{23}+146231240x^{56}y^{24}+328966456x^{55}y^{25}+698916700x^{54}y^{26}+1398331320x^{53}y^{27}+2648999020x^{52}y^{28}+4753314760x^{51}y^{29}+8074393848x^{50}y^{30}+13016369000x^{49}y^{31}+19929035890x^{48}y^{32}+28977892280x^{47}y^{33}+40051706600x^{46}y^{34}+52648574576x^{45}y^{35}+65817717640x^{44}y^{36}+78276434600x^{43}y^{37}+88592500160x^{42}y^{38}+95411915280x^{41}y^{39}+97791748000x^{40}y^{40}+95403790440x^{39}y^{41}+88576935160x^{38}y^{42}+78265921160x^{37}y^{43}+65811302880x^{36}y^{44}+52646621600x^{35}y^{45}+40061355180x^{34}y^{46}+28988386320x^{33}y^{47}+19935013920x^{32}y^{48}+13020784920x^{31}y^{49}+8071755924x^{30}y^{50}+4747023360x^{29}y^{51}+2646616500x^{28}y^{52}+1397829280x^{27}y^{53}+698391420x^{26}y^{54}+329156008x^{25}y^{55}+146666760x^{24}y^{56}+61903720x^{23}y^{57}+24747180x^{22}y^{58}+9373160x^{21}y^{59}+3288012x^{20}y^{60}+1114760x^{19}y^{61}+374880x^{18}y^{62}+115720x^{17}y^{63}+36485x^{16}y^{64}+9608x^{15}y^{65}+2880x^{14}y^{66}+880x^{13}y^{67}+320x^{12}y^{68}+40x^{11}y^{69}+40x^{10}y^{70}$

$[90, 45, 8]$	485.3411	$x^{90}y^0 + 90x^{82}y^8 + 135x^{81}y^9 + 360x^{80}y^{10} + 675x^{79}y^{11} + 1845x^{78}y^{12} + 4050x^{77}y^{13} + 10080x^{76}y^{14} + 23178x^{75}y^{15} + 58770x^{74}y^{16} + 139725x^{73}y^{17} + 345885x^{72}y^{18} + 861435x^{71}y^{19} + 2285064x^{70}y^{20} + 6194550x^{69}y^{21} + 16883280x^{68}y^{22} + 46867095x^{67}y^{23} + 126512835x^{66}y^{24} + 328254318x^{65}y^{25} + 817530435x^{64}y^{26} + 1941815670x^{63}y^{27} + 4377133125x^{62}y^{28} + 9376597215x^{61}y^{29} + 19113074226x^{60}y^{30} + 37033363305x^{59}y^{31} + 68305342050x^{58}y^{32} + 120095590275x^{57}y^{33} + 201305059965x^{56}y^{34} + 321969556089x^{55}y^{35} + 491798172190x^{54}y^{36} + 717619755600x^{53}y^{37} + 1000735356780x^{52}y^{38} + 1334314951815x^{51}y^{39} + 1701366130809x^{50}y^{40} + 2074981524750x^{49}y^{41} + 2421071879595x^{48}y^{42} + 2702839582350x^{47}y^{43} + 2887188120585x^{46}y^{44} + 2951331861745x^{45}y^{45} + 2887033258710x^{44}y^{46} + 2702492961165x^{43}y^{47} + 2420794467090x^{42}y^{48} + 2074846126275x^{41}y^{49} + 1701339849855x^{40}y^{50} + 1334470978845x^{39}y^{51} + 1000961372700x^{38}y^{52} + 717763300830x^{37}y^{53} + 491864034840x^{36}y^{54} + 321974740413x^{35}y^{55} + 201243891645x^{34}y^{56} + 120029840190x^{33}y^{57} + 68267765745x^{32}y^{58} + 36998278470x^{31}y^{59} + 19096186827x^{30}y^{60} + 9385276905x^{29}y^{61} + 4390550910x^{28}y^{62} + 1953347475x^{27}y^{63} + 825132825x^{26}y^{64} + 331427853x^{25}y^{65} + 127263975x^{24}y^{66} + 46729935x^{23}y^{67} + 16428510x^{22}y^{68} + 5464380x^{21}y^{69} + 1754604x^{20}y^{70} + 571365x^{19}y^{71} + 172685x^{18}y^{72} + 51615x^{17}y^{73} + 14265x^{16}y^{74} + 4275x^{15}y^{75} + 1170x^{14}y^{76} + 405x^{13}y^{77} + 90x^{12}y^{78} + 45x^{11}y^{79}$
$[100, 50, 8]$	991.8868	$x^{100}y^0 + 100x^{92}y^8 + 150x^{91}y^9 + 400x^{90}y^{10} + 750x^{89}y^{11} + 2050x^{88}y^{12} + 4500x^{87}y^{13} + 11200x^{86}y^{14} + 25750x^{85}y^{15} + 65550x^{84}y^{16} + 154700x^{83}y^{17} + 376475x^{82}y^{18} + 895950x^{81}y^{19} + 2175050x^{80}y^{20} + 5361800x^{79}y^{21} + 13466350x^{78}y^{22} + 34798200x^{77}y^{23} + 92208425x^{76}y^{24} + 248838520x^{75}y^{25} + 664041975x^{74}y^{26} + 1745854500x^{73}y^{27} + 4466465225x^{72}y^{28} + 10984937400x^{71}y^{29} + 25923313090x^{70}y^{30} + 58582091300x^{69}y^{31} + 126516128375x^{68}y^{32} + 261080661750x^{67}y^{33} + 515220406225x^{66}y^{34} + 972536877210x^{65}y^{35} + 1756705539825x^{64}y^{36} + 3039251321750x^{63}y^{37} + 5038795567600x^{62}y^{38} + 8008850277650x^{61}y^{39} + 12211230262005x^{60}y^{40} + 17867370854200x^{59}y^{41} + 25096115427375x^{58}y^{42} + 33848742673700x^{57}y^{43} + 43849967724625x^{56}y^{44} + 54571130333700x^{55}y^{45} + 65252414530150x^{54}y^{46} + 74976327517700x^{53}y^{47} + 82790608035525x^{52}y^{48} + 87861292423450x^{51}y^{49} + 89618026161551x^{50}y^{50} + 87857405478850x^{49}y^{51} + 82783905865625x^{48}y^{52} + 74970028200850x^{47}y^{53} + 65248611512600x^{46}y^{54} + 54570568359470x^{45}y^{55} + 43852363176475x^{44}y^{56} + 33852674178800x^{43}y^{57} + 25099888996525x^{42}y^{58} + 17869645949300x^{41}y^{59} + 12212100747595x^{40}y^{60} + 8008484200800x^{39}y^{61} + 5037395190350x^{38}y^{62} + 3037860252500x^{37}y^{63} + 1755687027300x^{36}y^{64} + 971925368250x^{35}y^{65} + 515169024275x^{34}y^{66} + 261301530550x^{33}y^{67} + 126753797475x^{32}y^{68} + 58793867450x^{31}y^{69} + 26080156040x^{30}y^{70} + 11071652750x^{29}y^{71} + 4495754275x^{28}y^{72} + 1745875850x^{27}y^{73} + 649815325x^{26}y^{74} + 232599062x^{25}y^{75} + 80888025x^{24}y^{76} + 26973200x^{23}y^{77} + 8640650x^{22}y^{78} + 2744750x^{21}y^{79} + 841105x^{20}y^{80} + 260650x^{19}y^{81} + 72050x^{18}y^{82} + 21600x^{17}y^{83} + 6025x^{16}y^{84} + 1750x^{15}y^{85} + 450x^{14}y^{86} + 250x^{13}y^{87}$

$[104, 52, 8]$	1318.4999	$x^{104}y^0 + 104x^{96}y^8 + 156x^{95}y^9 + 416x^{94}y^{10} + 780x^{93}y^{11} + 2132x^{92}y^{12} + 4680x^{91}y^{13} + 11648x^{90}y^{14} + 26780x^{89}y^{15} + 68380x^{88}y^{16} + 161460x^{87}y^{17} + 393432x^{86}y^{18} + 932204x^{85}y^{19} + 2259790x^{84}y^{20} + 5423444x^{83}y^{21} + 13312234x^{82}y^{22} + 33302932x^{81}y^{23} + 85420660x^{80}y^{24} + 223498392x^{79}y^{25} + 594318608x^{78}y^{26} + 1582023768x^{77}y^{27} + 4135034735x^{76}y^{28} + 10559623100x^{75}y^{29} + 26085750678x^{74}y^{30} + 61935626272x^{73}y^{31} + 141152315473x^{72}y^{32} + 308295054548x^{71}y^{33} + 644669532624x^{70}y^{34} + 1290987916348x^{69}y^{35} + 2477215357162x^{68}y^{36} + 4555925547628x^{67}y^{37} + 8035299821848x^{66}y^{38} + 13599981853228x^{65}y^{39} + 22098879922262x^{64}y^{40} + 34490126014656x^{63}y^{41} + 51727069528264x^{62}y^{42} + 74573090550972x^{61}y^{43} + 103374837758569x^{60}y^{44} + 137827877631340x^{59}y^{45} + 176781404663058x^{58}y^{46} + 218164385577008x^{57}y^{47} + 259085819420909x^{56}y^{48} + 296116440645428x^{55}y^{49} + 325741711177688x^{54}y^{50} + 344909846612292x^{53}y^{51} + 351540723574656x^{52}y^{52} + 344896297301140x^{51}y^{53} + 325720189289872x^{50}y^{54} + 296094551738492x^{49}y^{55} + 259071585082232x^{48}y^{56} + 218161303537216x^{47}y^{57} + 176787872762176x^{46}y^{58} + 137840919938436x^{45}y^{59} + 103388778374661x^{44}y^{60} + 74582544826452x^{43}y^{61} + 51731017854074x^{42}y^{62} + 34489026704080x^{41}y^{63} + 22094328412702x^{40}y^{64} + 13595100259212x^{39}y^{65} + 8031687015456x^{38}y^{66} + 4553635252244x^{37}y^{67} + 2476382714234x^{36}y^{68} + 1291342405444x^{35}y^{69} + 645476442416x^{34}y^{70} + 309144623476x^{33}y^{71} + 141844443078x^{32}y^{72} + 62314178004x^{31}y^{73} + 26228018856x^{30}y^{74} + 10593432616x^{29}y^{75} + 4105944635x^{28}y^{76} + 1529845564x^{27}y^{77} + 546808862x^{26}y^{78} + 188697028x^{25}y^{79} + 63663015x^{24}y^{80} + 20707128x^{23}y^{81} + 6529120x^{22}y^{82} + 2020096x^{21}y^{83} + 623662x^{20}y^{84} + 176488x^{19}y^{85} + 56446x^{18}y^{86} + 13520x^{17}y^{87} + 3952x^{16}y^{88} + 1144x^{15}y^{89} + 416x^{14}y^{90} + 52x^{13}y^{91} + 52x^{12}y^{92}$

$[108, 54, 8]$	1751.5857	$x^{108}y^0 + 108x^{100}y^8 + 162x^{99}y^9 + 432x^{98}y^{10} + 810x^{97}y^{11} + 2214x^{96}y^{12} + 4860x^{95}y^{13} + 12096x^{94}y^{14} + 27810x^{93}y^{15} + 71226x^{92}y^{16} + 168318x^{91}y^{17} + 410658x^{90}y^{18} + 972810x^{89}y^{19} + 2351862x^{88}y^{20} + 5609790x^{87}y^{21} + 13627494x^{86}y^{22} + 32962680x^{85}y^{23} + 82551348x^{84}y^{24} + 208678572x^{83}y^{25} + 544673970x^{82}y^{26} + 1424653290x^{81}y^{27} + 3771633564x^{80}y^{28} + 9815955192x^{79}y^{29} + 25033107915x^{78}y^{30} + 61943965326x^{77}y^{31} + 147956700303x^{76}y^{32} + 339812169174x^{75}y^{33} + 749412463023x^{74}y^{34} + 1586457488844x^{73}y^{35} + 3220506549780x^{72}y^{36} + 6274038404790x^{71}y^{37} + 11732790198726x^{70}y^{38} + 21069156489300x^{69}y^{39} + 36353414343474x^{68}y^{40} + 60296405355396x^{67}y^{41} + 96181022720826x^{66}y^{42} + 147604875097410x^{65}y^{43} + 218022874472439x^{64}y^{44} + 310042292199768x^{63}y^{45} + 424586681117487x^{62}y^{46} + 560081101245228x^{61}y^{47} + 711776640727035x^{60}y^{48} + 871595979232374x^{59}y^{49} + 1028540563040529x^{58}y^{50} + 1169770589224650x^{57}y^{51} + 1282299490199970x^{56}y^{52} + 1354903707705462x^{55}y^{53} + 1379987622460288x^{54}y^{54} + 1354859428077036x^{53}y^{55} + 1282222287590616x^{52}y^{56} + 1169697763800612x^{51}y^{57} + 1028485208314800x^{50}y^{58} + 871583347557450x^{49}y^{59} + 711798606433881x^{48}y^{60} + 560121775271676x^{47}y^{61} + 424637107824087x^{46}y^{62} + 310075234411884x^{45}y^{63} + 218040323455152x^{44}y^{64} + 147605545891674x^{43}y^{65} + 96167492502921x^{42}y^{66} + 60280096062150x^{41}y^{67} + 36337937391000x^{40}y^{68} + 21059670928122x^{39}y^{69} + 11728660753926x^{38}y^{70} + 6274617980364x^{37}y^{71} + 3223270669662x^{36}y^{72} + 1589130135090x^{35}y^{73} + 752050480554x^{34}y^{74} + 341451396108x^{33}y^{75} + 148828542939x^{32}y^{76} + 62235587412x^{31}y^{77} + 24973610265x^{30}y^{78} + 9651105666x^{29}y^{79} + 3584656107x^{28}y^{80} + 1291394972x^{27}y^{81} + 445249845x^{26}y^{82} + 150021396x^{25}y^{83} + 48595356x^{24}y^{84} + 15848028x^{23}y^{85} + 4766958x^{22}y^{86} + 1509372x^{21}y^{87} + 430488x^{20}y^{88} + 136728x^{19}y^{89} + 33642x^{18}y^{90} + 11988x^{17}y^{91} + 1971x^{16}y^{92} + 1188x^{15}y^{93} + 54x^{14}y^{94} + 54x^{13}y^{95} + 6x^9y^{99}$

B.5 Weight enumerator for constraint length $m = 6$

Secrecy gain: Constraint length 6		
$[n, k, d]$	Secrecy gain	$W_{\mathscr{C}}(x, y)$
$[12, 6, 4]$	1.6569	$x^{12}y^0 + 6x^8y^4 + 24x^7y^5 + 16x^6y^6 + 9x^4y^8 + 8x^3y^9$
$[14, 7, 4]$	1.8283	$x^{14}y^0 + 7x^{10}y^4 + 21x^9y^5 + 21x^8y^6 + 29x^7y^7 + 28x^6y^8 + 7x^5y^9 + 7x^4y^{10} + 7x^3y^{11}$
$[16, 8, 5]$	2.1410	$x^{16}y^0 + 24x^{11}y^5 + 44x^{10}y^6 + 40x^9y^7 + 45x^8y^8 + 40x^7y^9 + 28x^6y^{10} + 24x^5y^{11} + 10x^4y^{12}$
$[18, 9, 6]$	2.4854	$x^{18}y^0 + 102x^{12}y^6 + 153x^{10}y^8 + 153x^8y^{10} + 102x^6y^{12} + 1x^0y^{18}$
$[20, 10, 6]$	2.8132	$x^{20}y^0 + 90x^{14}y^6 + 255x^{12}y^8 + 332x^{10}y^{10} + 255x^8y^{12} + 90x^6y^{14} + 1x^0y^{20}$
$[22, 11, 7]$	3.3347	$x^{22}y^0 + 176x^{15}y^7 + 330x^{14}y^8 + 672x^{11}y^{11} + 616x^{10}y^{12} + 176x^7y^{15} + 77x^6y^{16}$
$[24, 12, 8]$	3.8788	$x^{24}y^0 + 759x^{16}y^8 + 2576x^{12}y^{12} + 759x^8y^{16} + 1x^0y^{24}$
$[26, 13, 7]$	4.3555	$x^{26}y^0 + 117x^{19}y^7 + 273x^{18}y^8 + 338x^{17}y^9 + 598x^{16}y^{10} + 923x^{15}y^{11} + 1105x^{14}y^{12} + 1340x^{13}y^{13} + 1300x^{12}y^{14} + 923x^{11}y^{15} + 598x^{10}y^{16} + 338x^9y^{17} + 182x^8y^{18} + 117x^7y^{19} + 39x^6y^{20}$
$[28, 14, 8]$	5.0819	$x^{28}y^0 + 546x^{20}y^8 + 1456x^{18}y^{10} + 3549x^{16}y^{12} + 5280x^{14}y^{14} + 3549x^{12}y^{16} + 1456x^{10}y^{18} + 546x^8y^{20} + 1x^0y^{28}$
$[30, 15, 8]$	5.8431	$x^{30}y^0 + 450x^{22}y^8 + 1848x^{20}y^{10} + 5040x^{18}y^{12} + 9045x^{16}y^{14} + 9045x^{14}y^{16} + 5040x^{12}y^{18} + 1848x^{10}y^{20} + 450x^8y^{22} + 1x^0y^{30}$
$[32, 16, 8]$	6.7258	$x^{32}y^0 + 380x^{24}y^8 + 1920x^{22}y^{10} + 7168x^{20}y^{12} + 13440x^{18}y^{14} + 19718x^{16}y^{16} + 13440x^{14}y^{18} + 7168x^{12}y^{20} + 1920x^{10}y^{22} + 380x^8y^{24} + 1x^0y^{32}$
$[34, 17, 8]$	7.7714	$x^{34}y^0 + 306x^{26}y^8 + 1972x^{24}y^{10} + 8636x^{22}y^{12} + 20604x^{20}y^{14} + 34017x^{18}y^{16} + 34017x^{16}y^{18} + 20604x^{14}y^{20} + 8636x^{12}y^{22} + 1972x^{10}y^{24} + 306x^8y^{26} + 1x^0y^{34}$
$[36, 18, 8]$	9.0220	$x^{36}y^0 + 225x^{28}y^8 + 2016x^{26}y^{10} + 9555x^{24}y^{12} + 28800x^{22}y^{14} + 55755x^{20}y^{16} + 69440x^{18}y^{18} + 55755x^{16}y^{20} + 28800x^{14}y^{22} + 9555x^{12}y^{24} + 2016x^{10}y^{26} + 225x^8y^{28} + 1x^0y^{36}$
$[38, 19, 8]$	10.44310	$x^{38}y^0 + 190x^{30}y^8 + 1767x^{28}y^{10} + 10507x^{26}y^{12} + 36860x^{24}y^{14} + 84341x^{22}y^{16} + 128478x^{20}y^{18} + 128478x^{18}y^{20} + 84341x^{16}y^{22} + 36860x^{14}y^{24} + 10507x^{12}y^{26} + 1767x^{10}y^{28} + 190x^8y^{30} + 1x^0y^{38}$
$[40, 20, 9]$	12.4028	$x^{40}y^0 + 280x^{31}y^9 + 1010x^{30}y^{10} + 2420x^{29}y^{11} + 5425x^{28}y^{12} + 10880x^{27}y^{13} + 21200x^{26}y^{14} + 38824x^{25}y^{15} + 61115x^{24}y^{16} + 84740x^{23}y^{17} + 107980x^{22}y^{18} + 124920x^{21}y^{19} + 130690x^{20}y^{20} + 125360x^{19}y^{21} + 108520x^{18}y^{22} + 84320x^{17}y^{23} + 60210x^{16}y^{24} + 38704x^{15}y^{25} + 21970x^{14}y^{26} + 11380x^{13}y^{27} + 5165x^{12}y^{28} + 2160x^{11}y^{29} + 952x^{10}y^{30} + 280x^9y^{31} + 50x^8y^{32} + 20x^7y^{33}$
$[42, 21, 9]$	14.3805	$x^{42}y^0 + 259x^{33}y^9 + 735x^{32}y^{10} + 2058x^{31}y^{11} + 5761x^{30}y^{12} + 11886x^{29}y^{13} + 24255x^{28}y^{14} + 47530x^{27}y^{15} + 79569x^{26}y^{16} + 121233x^{25}y^{17} + 169512x^{24}y^{18} + 212436x^{23}y^{19} + 244650x^{22}y^{20} + 258164x^{21}y^{21} + 244398x^{20}y^{22} + 211428x^{19}y^{23} + 169078x^{18}y^{24} + 121989x^{17}y^{25} + 79359x^{16}y^{26} + 47586x^{15}y^{27} + 25413x^{14}y^{28} + 11550x^{13}y^{29} + 4963x^{12}y^{30} + 2226x^{11}y^{31} + 840x^{10}y^{32} + 231x^9y^{33} + 42x^8y^{34}$
$[44, 22, 10]$	16.8268	$x^{44}y^0 + 1364x^{34}y^{10} + 10109x^{32}y^{12} + 55176x^{30}y^{14} + 196218x^{28}y^{16} + 493548x^{26}y^{18} + 842248x^{24}y^{20} + 996976x^{22}y^{22} + 842248x^{20}y^{24} + 493548x^{18}y^{26} + 196218x^{16}y^{28} + 55176x^{14}y^{30} + 10109x^{12}y^{32} + 1364x^{10}y^{34} + 1x^0y^{44}$
$[46, 23, 10]$	19.6065	$x^{46}y^0 + 989x^{36}y^{10} + 9821x^{34}y^{12} + 56511x^{32}y^{14} + 235612x^{30}y^{16} + 673555x^{28}y^{18} + 1337427x^{26}y^{20} + 1880388x^{24}y^{22} + 1880388x^{22}y^{24} + 1337427x^{20}y^{26} + 673555x^{18}y^{28} + 235612x^{16}y^{30} + 56511x^{14}y^{32} + 9821x^{12}y^{34} + 989x^{10}y^{36} + 1x^0y^{46}$

$[48, 24, 9]$	22.6636	$x^{48}y^0 + 128x^{39}y^9 + 396x^{38}y^{10} + 1296x^{37}y^{11} + 4016x^{36}y^{12} + 11472x^{35}y^{13} + 29508x^{34}y^{14} + 65264x^{33}y^{15} + 133701x^{32}y^{16} + 253512x^{31}y^{17} + 435256x^{30}y^{18} + 685800x^{29}y^{19} + 996558x^{28}y^{20} + 1333096x^{27}y^{21} + 1636896x^{26}y^{22} + 1843896x^{25}y^{23} + 1918725x^{24}y^{24} + 1843536x^{23}y^{25} + 1630428x^{22}y^{26} + 1332896x^{21}y^{27} + 1000692x^{20}y^{28} + 686112x^{19}y^{29} + 434948x^{18}y^{30} + 253536x^{17}y^{31} + 134226x^{16}y^{32} + 65128x^{15}y^{33} + 28512x^{14}y^{34} + 11496x^{13}y^{35} + 4334x^{12}y^{36} + 1320x^{11}y^{37} + 408x^{10}y^{38} + 120x^9y^{39} + 3x^8y^{40}$
$[50, 25, 10]$	26.6343	$x^{50}y^0 + 575x^{40}y^{10} + 7400x^{38}y^{12} + 56450x^{36}y^{14} + 291525x^{34}y^{16} + 1079050x^{32}y^{18} + 2803490x^{30}y^{20} + 5298900x^{28}y^{22} + 7239825x^{26}y^{24} + 7239825x^{24}y^{26} + 5298900x^{22}y^{28} + 2803490x^{20}y^{30} + 1079050x^{18}y^{32} + 291525x^{16}y^{34} + 56450x^{14}y^{36} + 7400x^{12}y^{38} + 575x^{10}y^{40} + 1x^0y^{50}$
$[52, 26, 10]$	31.0259	$x^{52}y^0 + 494x^{42}y^{10} + 5993x^{40}y^{12} + 52910x^{38}y^{14} + 310557x^{36}y^{16} + 1268020x^{34}y^{18} + 3754270x^{32}y^{20} + 8068658x^{30}y^{22} + 12706395x^{28}y^{24} + 14774268x^{26}y^{26} + 12706395x^{24}y^{28} + 8068658x^{22}y^{30} + 3754270x^{20}y^{32} + 1268020x^{18}y^{34} + 310557x^{16}y^{36} + 52910x^{14}y^{38} + 5993x^{12}y^{40} + 494x^{10}y^{42} + 1x^0y^{52}$
$[54, 27, 10]$	36.2326	$x^{54}y^0 + 378x^{44}y^{10} + 4968x^{42}y^{12} + 48222x^{40}y^{14} + 316332x^{38}y^{16} + 1442442x^{36}y^{18} + 4785102x^{34}y^{20} + 11635731x^{32}y^{22} + 20904930x^{30}y^{24} + 27970758x^{28}y^{26} + 27970758x^{26}y^{28} + 20904930x^{24}y^{30} + 11635731x^{22}y^{32} + 4785102x^{20}y^{34} + 1442442x^{18}y^{36} + 316332x^{16}y^{38} + 48222x^{14}y^{40} + 4968x^{12}y^{42} + 378x^{10}y^{44} + 1x^0y^{54}$
$[56, 28, 10]$	42.1094	$x^{56}y^0 + 420x^{46}y^{10} + 3850x^{44}y^{12} + 41444x^{42}y^{14} + 315658x^{40}y^{16} + 1583344x^{38}y^{18} + 5838392x^{36}y^{20} + 15968624x^{34}y^{22} + 32464117x^{32}y^{24} + 49515032x^{30}y^{26} + 56973692x^{28}y^{28} + 49515032x^{26}y^{30} + 32464117x^{24}y^{32} + 15968624x^{22}y^{34} + 5838392x^{20}y^{36} + 1583344x^{18}y^{38} + 315658x^{16}y^{40} + 41444x^{14}y^{42} + 3850x^{12}y^{44} + 420x^{10}y^{46} + 1x^0y^{56}$
$[58, 29, 10]$	49.2791	$x^{58}y^0 + 319x^{48}y^{10} + 3103x^{46}y^{12} + 36772x^{44}y^{14} + 298700x^{42}y^{16} + 1684001x^{40}y^{18} + 6868911x^{38}y^{20} + 20944757x^{36}y^{22} + 47788723x^{34}y^{24} + 82572512x^{32}y^{26} + 108237657x^{30}y^{28} + 108237657x^{28}y^{30} + 82572512x^{26}y^{32} + 47788723x^{24}y^{34} + 20944757x^{22}y^{36} + 6868911x^{20}y^{38} + 1684001x^{18}y^{40} + 298700x^{16}y^{42} + 36772x^{14}y^{44} + 3103x^{12}y^{46} + 319x^{10}y^{48} + 1x^0y^{58}$
$[60, 30, 10]$	57.1035	$x^{60}y^0 + 420x^{50}y^{10} + 2595x^{48}y^{12} + 29220x^{46}y^{14} + 278505x^{44}y^{16} + 1741140x^{42}y^{18} + 7793640x^{40}y^{20} + 26349360x^{38}y^{22} + 67135320x^{36}y^{24} + 130235400x^{34}y^{26} + 193225395x^{32}y^{28} + 220159832x^{30}y^{30} + 193225395x^{28}y^{32} + 130235400x^{26}y^{34} + 67135320x^{24}y^{36} + 26349360x^{22}y^{38} + 7793640x^{20}y^{40} + 1741140x^{18}y^{42} + 278505x^{16}y^{44} + 29220x^{14}y^{46} + 2595x^{12}y^{48} + 420x^{10}y^{50} + 1x^0y^{60}$
$[70, 35, 9]$	123.2624	$x^{70}y^0 + 35x^{61}y^9 + 210x^{60}y^{10} + 385x^{59}y^{11} + 420x^{58}y^{12} + 2240x^{57}y^{13} + 6475x^{56}y^{14} + 20090x^{55}y^{15} + 71995x^{54}y^{16} + 222460x^{53}y^{17} + 659890x^{52}y^{18} + 1840755x^{51}y^{19} + 4707227x^{50}y^{20} + 11229685x^{49}y^{21} + 25034205x^{48}y^{22} + 52204460x^{47}y^{23} + 102200910x^{46}y^{24} + 187930211x^{45}y^{25} + 325027885x^{44}y^{26} + 529570195x^{43}y^{27} + 813570140x^{42}y^{28} + 1178901500x^{41}y^{29} + 1611155686x^{40}y^{30} + 2078627460x^{39}y^{31} + 2533646815x^{38}y^{32} + 2917036920x^{37}y^{33} + 3173826320x^{36}y^{34} + 3264929954x^{35}y^{35} + 3173870210x^{34}y^{36} + 2916535790x^{33}y^{37} + 2533800290x^{32}y^{38} + 2078903960x^{31}y^{39} + 1610901628x^{30}y^{40} + 1178858065x^{29}y^{41} + 813605600x^{28}y^{42} + 529680795x^{27}y^{43} + 325177160x^{26}y^{44} + 187836572x^{25}y^{45} + 102131295x^{24}y^{46} + 52231410x^{23}y^{47} + 25038405x^{22}y^{48} + 11217260x^{21}y^{49} + 4679542x^{20}y^{50} + 1837395x^{19}y^{51} + 677915x^{18}y^{52} + 228165x^{17}y^{53} + 72625x^{16}y^{54} + 22092x^{15}y^{55} + 6230x^{14}y^{56} + 1225x^{13}y^{57} + 105x^{12}y^{58} + 105x^{11}y^{59}$

$[80, 40, 9]$	263.5837	$x^{80}y^0 + 40x^{71}y^9 + 240x^{70}y^{10} + 440x^{69}y^{11} + 480x^{68}y^{12} + 1800x^{67}y^{13} + 4840x^{66}y^{14} + 11640x^{65}y^{15} + 34740x^{64}y^{16} + 104680x^{63}y^{17} + 329160x^{62}y^{18} + 1053080x^{61}y^{19} + 3161172x^{60}y^{20} + 9045840x^{59}y^{21} + 24467560x^{58}y^{22} + 61827280x^{57}y^{23} + 147402605x^{56}y^{24} + 331053624x^{55}y^{25} + 699873740x^{54}y^{26} + 1399526680x^{53}y^{27} + 2648786650x^{52}y^{28} + 4746349680x^{51}y^{29} + 8066410876x^{50}y^{30} + 13009986960x^{49}y^{31} + 19922317215x^{48}y^{32} + 28982893280x^{47}y^{33} + 40070521720x^{46}y^{34} + 52666877520x^{45}y^{35} + 65832344560x^{44}y^{36} + 78283172160x^{43}y^{37} + 88579181760x^{42}y^{38} + 95385747840x^{41}y^{39} + 97767318282x^{40}y^{40} + 95387608440x^{39}y^{41} + 88577129880x^{38}y^{42} + 78281938440x^{37}y^{43} + 65834327820x^{36}y^{44} + 52665164680x^{35}y^{45} + 40066831600x^{34}y^{46} + 28983973240x^{33}y^{47} + 19924853790x^{32}y^{48} + 13010790600x^{31}y^{49} + 8067969480x^{30}y^{50} + 4746934680x^{29}y^{51} + 2647255500x^{28}y^{52} + 1398716080x^{27}y^{53} + 699759480x^{26}y^{54} + 330748784x^{25}y^{55} + 147379145x^{24}y^{56} + 61936360x^{23}y^{57} + 24573580x^{22}y^{58} + 9189480x^{21}y^{59} + 3225642x^{20}y^{60} + 1063360x^{19}y^{61} + 327180x^{18}y^{62} + 90560x^{17}y^{63} + 23630x^{16}y^{64} + 6320x^{15}y^{65} + 1560x^{14}y^{66} + 320x^{13}y^{67}$
$[90, 45, 9]$	560.9149	$x^{90}y^0+90x^{81}y^9+180x^{80}y^{10}+405x^{79}y^{11}+675x^{78}y^{12}+1800x^{77}y^{13}+5265x^{76}y^{14} + 11355x^{75}y^{15} + 27945x^{74}y^{16} + 71460x^{73}y^{17} + 193920x^{72}y^{18} + 568845x^{71}y^{19} + 1692549x^{70}y^{20} + 5127255x^{69}y^{21} + 15408315x^{68}y^{22} + 44714070x^{67}y^{23} + 124255680x^{66}y^{24} + 327906549x^{65}y^{25} + 820671795x^{64}y^{26} + 1950388415x^{63}y^{27} + 4397063940x^{62}y^{28} + 9409018140x^{61}y^{29} + 19140001746x^{60}y^{30} + 37048839525x^{59}y^{31} + 68302262430x^{58}y^{32} + 120025046520x^{57}y^{33} + 201175173855x^{56}y^{34} + 321833604501x^{55}y^{35} + 491664560060x^{54}y^{36} + 717575277555x^{53}y^{37} + 1000896898050x^{52}y^{38} + 1334618980620x^{51}y^{39} + 1701703574973x^{50}y^{40} + 2075289435765x^{49}y^{41} + 2421151476060x^{48}y^{42} + 2702599920945x^{47}y^{43} + 2886795754155x^{46}y^{44} + 2950873996824x^{45}y^{45} + 2886661278405x^{44}y^{46} + 2702423855925x^{43}y^{47} + 2420982430305x^{42}y^{48} + 2075177045340x^{41}y^{49} + 1701699947118x^{40}y^{50} + 1334698412895x^{39}y^{51} + 1001009231685x^{38}y^{52} + 717687020205x^{37}y^{53} + 491735445655x^{36}y^{54} + 321841401894x^{35}y^{55} + 201139335810x^{34}y^{56}+119977903575x^{33}y^{57}+68261808555x^{32}y^{58}+37024985565x^{31}y^{59} + 19135429854x^{30}y^{60} + 9414274500x^{29}y^{61} + 4404385800x^{28}y^{62} + 1958297195x^{27}y^{63} + 826607655x^{26}y^{64} + 330443784x^{25}y^{65} + 124805955x^{24}y^{66} + 44494335x^{23}y^{67} + 14903370x^{22}y^{68}+4628745x^{21}y^{69}+1323900x^{20}y^{70}+351000x^{19}y^{71}+85425x^{18}y^{72} + 18405x^{17}y^{73} + 3330x^{16}y^{74} + 414x^{15}y^{75}$

$[100, 50, 8]$	1191.6311	$x^{100}y^0 + 50x^{92}y^8 + 250x^{90}y^{10} + 1750x^{88}y^{12} + 9400x^{86}y^{14} + 54975x^{84}y^{16} + 332500x^{82}y^{18} + 2268995x^{80}y^{20} + 18030200x^{78}y^{22} + 157491550x^{76}y^{24} + 1274835250x^{74}y^{26} + 8906295050x^{72}y^{28} + 52101170400x^{70}y^{30} + 253684672250x^{68}y^{32} + 1030996946800x^{66}y^{34} + 3512343377525x^{64}y^{36} + 10073884380000x^{62}y^{38} + 24420739141860x^{60}y^{40} + 50197167276500x^{58}y^{42} + 87707861712300x^{56}y^{44} + 130505742560400x^{54}y^{46} + 165571281694350x^{52}y^{48} + 179227582337912x^{50}y^{50} + 165571281694350x^{48}y^{52} + 130505742560400x^{46}y^{54} + 87707861712300x^{44}y^{56} + 50197167276500x^{42}y^{58} + 24420739141860x^{40}y^{60} + 10073884380000x^{38}y^{62} + 3512343377525x^{36}y^{64} + 1030996946800x^{34}y^{66} + 253684672250x^{32}y^{68} + 52101170400x^{30}y^{70} + 8906295050x^{28}y^{72} + 1274835250x^{26}y^{74} + 157491550x^{24}y^{76} + 18030200x^{22}y^{78} + 2268995x^{20}y^{80} + 332500x^{18}y^{82} + 54975x^{16}y^{84} + 9400x^{14}y^{86} + 1750x^{12}y^{88} + 250x^{10}y^{90} + 50x^8y^{92} + 1x^0y^{100}$
$[104, 52, 8]$	1609.7222	$x^{104}y^0 + 52x^{96}y^8 + 260x^{94}y^{10} + 1820x^{92}y^{12} + 9724x^{90}y^{14} + 57252x^{88}y^{16} + 335972x^{86}y^{18} + 2117206x^{84}y^{20} + 15421796x^{82}y^{22} + 127567648x^{80}y^{24} + 1069489932x^{78}y^{26} + 8081913788x^{76}y^{28} + 52189990580x^{74}y^{30} + 283427590498x^{72}y^{32} + 1291847929840x^{70}y^{34} + 4955796754792x^{68}y^{36} + 16065661346800x^{66}y^{38} + 44184912826769x^{64}y^{40} + 103449361895048x^{62}y^{42} + 206768557952424x^{60}y^{44} + 353589899385528x^{58}y^{46} + 518173123491348x^{56}y^{48} + 651449861037144x^{54}y^{50} + 703051753138052x^{52}y^{52} + 651449861037144x^{50}y^{54} + 518173123491348x^{48}y^{56} + 353589899385528x^{46}y^{58} + 206768557952424x^{44}y^{60} + 103449361895048x^{42}y^{62} + 44184912826769x^{40}y^{64} + 16065661346800x^{38}y^{66} + 4955796754792x^{36}y^{68} + 1291847929840x^{34}y^{70} + 283427590498x^{32}y^{72} + 52189990580x^{30}y^{74} + 8081913788x^{28}y^{76} + 1069489932x^{26}y^{78} + 127567648x^{24}y^{80} + 15421796x^{22}y^{82} + 2117206x^{20}y^{84} + 335972x^{18}y^{86} + 57252x^{16}y^{88} + 9724x^{14}y^{90} + 1820x^{12}y^{92} + 260x^{10}y^{94} + 52x^8y^{96} + 1x^0y^{104}$
$[108, 54, 8]$	2173.2995	$x^{108}y^0 + 54x^{100}y^8 + 270x^{98}y^{10} + 1890x^{96}y^{12} + 10098x^{94}y^{14} + 59049x^{92}y^{16} + 346482x^{90}y^{18} + 2085426x^{88}y^{20} + 13843116x^{86}y^{22} + 106278570x^{84}y^{24} + 883309914x^{82}y^{26} + 7031560293x^{80}y^{28} + 49419952650x^{78}y^{30} + 296465485428x^{76}y^{32} + 1503833930406x^{74}y^{34} + 6450500013918x^{72}y^{36} + 23467887151632x^{70}y^{38} + 72684550144332x^{68}y^{40} + 192317157732780x^{66}y^{42} + 436034491273305x^{64}y^{44} + 849246542930052x^{62}y^{46} + 1423648758659778x^{60}y^{48} + 2057078421638100x^{58}y^{50} + 2564477721808452x^{56}y^{52} + 2759870933049992x^{54}y^{54} + 2564477721808452x^{52}y^{56} + 2057078421638100x^{50}y^{58} + 1423648758659778x^{48}y^{60} + 849246542930052x^{46}y^{62} + 436034491273305x^{44}y^{64} + 192317157732780x^{42}y^{66} + 72684550144332x^{40}y^{68} + 23467887151632x^{38}y^{70} + 6450500013918x^{36}y^{72} + 1503833930406x^{34}y^{74} + 296465485428x^{32}y^{76} + 49419952650x^{30}y^{78} + 7031560293x^{28}y^{80} + 883309914x^{26}y^{82} + 106278570x^{24}y^{84} + 13843116x^{22}y^{86} + 2085426x^{20}y^{88} + 346482x^{18}y^{90} + 59049x^{16}y^{92} + 10098x^{14}y^{94} + 1890x^{12}y^{96} + 270x^{10}y^{98} + 54x^8y^{100} + 1x^0y^{108}$