# 2
# Transmission Concept

A super block is one symbol, which contains a number of blocks. Each block contains a number of time slots. For each super block length in blocks, we calculate
1. Number of ways to organize the blocks in a super block

$$n!$$

2. Number of Bits / Symbol

$$log_2(n!)$$

3. Number of Bits / Photon

$$\frac{log_2(n!)}{n}$$

4. Number of Bits / Time Slot

$$\frac{log_2(n!)}{n} \times \frac{n}{T}$$

For example, we have [1,2,4,7], there are 4! = 24 permutation of ways to organize the blocks to generate different super blocks representing the corresponding symbols as follow:

$$[1,2,4,7] \to A, [1,2,7,4] \to B, [1,4,2,7] \to C, [1,4,7,2] \to D, [1,7,2,4] \to E, [1,7,4,2] \to F$$

$$[2,1,4,7] \to G, [2,1,7,4] \to H, [2,4,1,7] \to I, [2,4,7,1] \to J, [2,7,1,4] \to K, [2,7,4,1] \to L$$

$$[4,1,2,7] \to M, [4,1,7,2] \to N, [4,2,1,7] \to O, [4,2,7,1] \to P, [4,7,1,2] \to Q, [4,7,2,1] \to R$$

$$[7,1,2,4] \to S, [7,1,4,2] \to T, [7,2,1,4] \to U, [7,2,4,1] \to V, [7,4,1,2] \to W, [7,4,2,1] \to X$$

The information content of the super block is

$$log_2(4!) = 4.6 \quad \text{bits/symbol}$$

For each photon, it contains

$$1.15 \quad \text{bits/photon}$$

For each time slot, it has

$$0.33 bits/timeslot$$

# 3
# Our Method

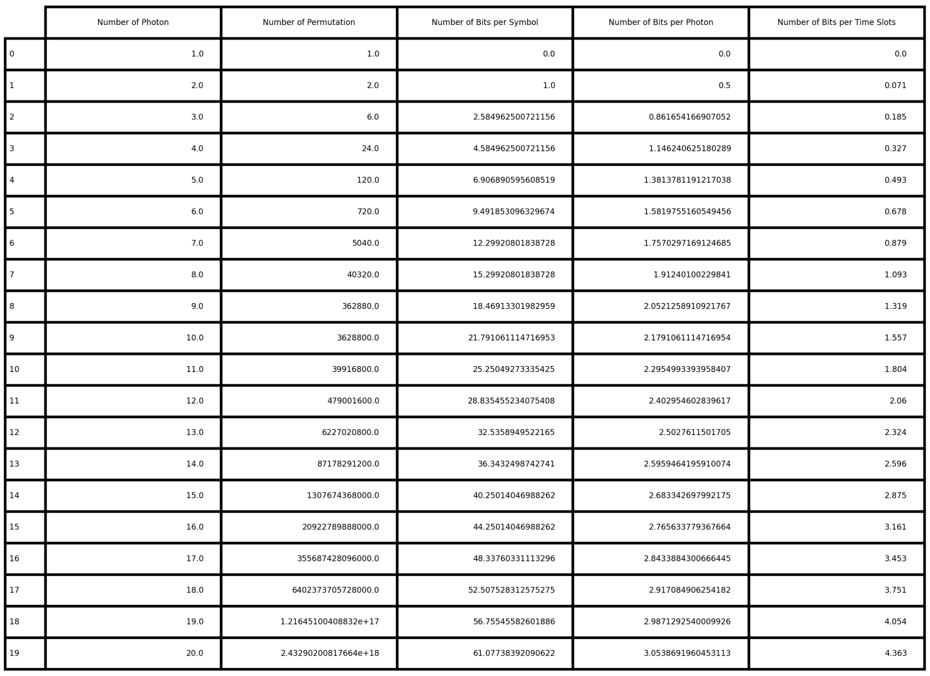
We define a block as an integral number of time bins ( or time slots, or other encoding sources that are orthogonal, i.e., that can be perfectly discriminated).

We represent [1,2,4,7] into a form with polarization of photon:

1 |2 |4 |7
H H0 H000 V000000, which comprises of 14 lengths of block, equivalent to 14 time slots.

Our method uses 4 photons in 14 time slots. It means that our method has:
1. 4! = 24 ways to order them
2. 4.6 bits/symbol
3. 1.15 bits/photon
4. 0.33 bits / time slot

We can check our values form in the 5th iteration of the following table.



| | Number of Photon | Number of Permutation | Number of Bits per Symbol | Number of Bits per Photon | Number of Bits per Time Slots |
|---|---|---|---|---|---|
| 0 | 1.0 | 1.0 | 0.0 | 0.0 | 0.0 |
| 1 | 2.0 | 2.0 | 1.0 | 0.5 | 0.071 |
| 2 | 3.0 | 6.0 | 2.584962500721156 | 0.861654166907052 | 0.185 |
| 3 | 4.0 | 24.0 | 4.584962500721156 | 1.146240625180289 | 0.327 |
| 4 | 5.0 | 120.0 | 6.906890595608519 | 1.3813781191217038 | 0.493 |
| 5 | 6.0 | 720.0 | 9.491853096329674 | 1.5819755160549456 | 0.678 |
| 6 | 7.0 | 5040.0 | 12.29920801838728 | 1.7570297169124685 | 0.879 |
| 7 | 8.0 | 40320.0 | 15.29920801838728 | 1.91240100229841 | 1.093 |
| 8 | 9.0 | 362880.0 | 18.46913301982959 | 2.0521258910921767 | 1.319 |
| 9 | 10.0 | 3628800.0 | 21.791061114716953 | 2.1791061114716954 | 1.557 |
| 10 | 11.0 | 39916800.0 | 25.25049273335425 | 2.2954993393958407 | 1.804 |
| 11 | 12.0 | 479001600.0 | 28.835455234075408 | 2.402954602839617 | 2.06 |
| 12 | 13.0 | 6227020800.0 | 32.5358949522165 | 2.5027611501705 | 2.324 |
| 13 | 14.0 | 87178291200.0 | 36.3432498742741 | 2.5959464195910074 | 2.596 |
| 14 | 15.0 | 1307674368000.0 | 40.25014046988262 | 2.683342697992175 | 2.875 |
| 15 | 16.0 | 20922789888000.0 | 44.25014046988262 | 2.765633779367664 | 3.161 |
| 16 | 17.0 | 355687428096000.0 | 48.33760331113296 | 2.8433884300666445 | 3.453 |
| 17 | 18.0 | 6402373705728000.0 | 52.507528312575275 | 2.917084906254182 | 3.751 |
| 18 | 19.0 | 1.21645100408832e+17 | 56.75545582601886 | 2.9871292540009926 | 4.054 |
| 19 | 20.0 | 2.43290200817664e+18 | 61.07738392090622 | 3.0538691960453113 | 4.363 |

**Figure 3.1:** Table

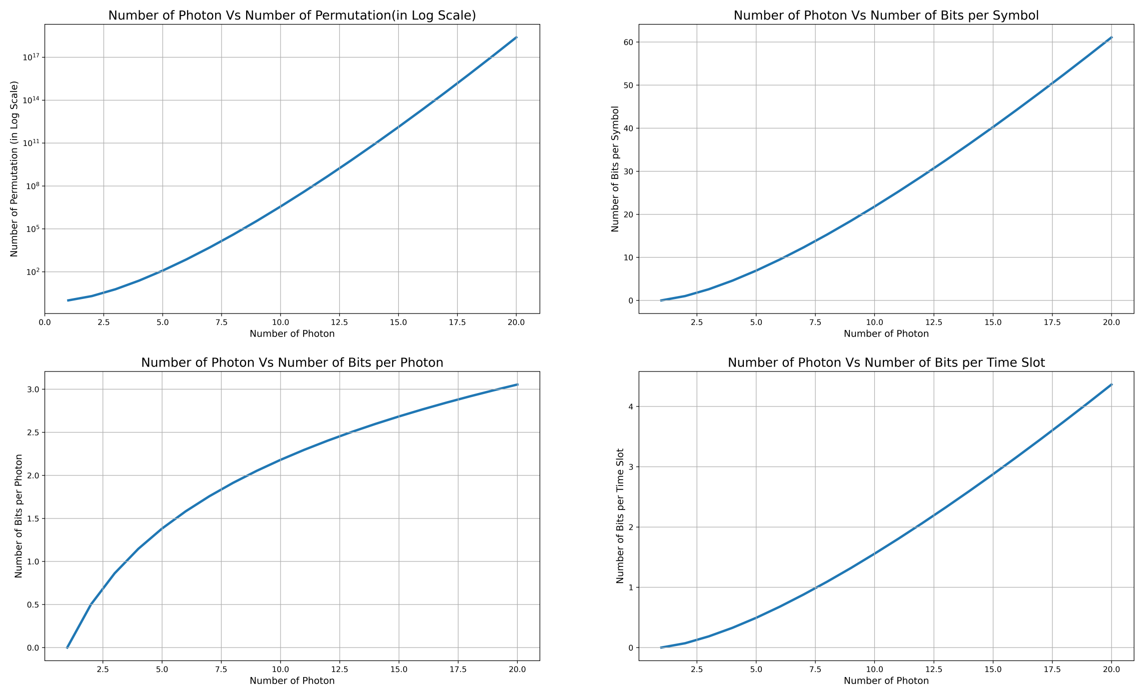


**Figure 3.2:** Number of Photons Vs Number of Permutation, Number of Photons Vs Number of Bits per Symbol, Number of Photons Vs Number of Bits per Photon, Number of Photons Vs Number of Bits per Time Slot

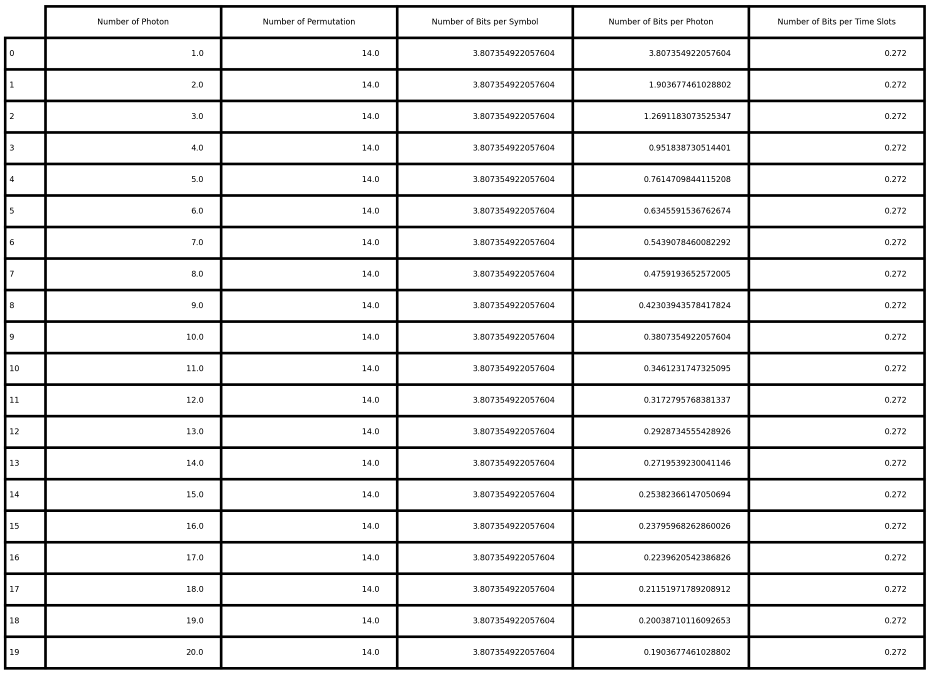
# 4

# PPM

The representation of bits to symbol is as follow:
100000 $\rightarrow A$
010000 $\rightarrow B$
001000 $\rightarrow C$
000100 $\rightarrow D$
000010 $\rightarrow E$
000001 $\rightarrow F$

PPM uses 1 photon in 14 time slots
It means that PPM has:
1. 14 ways to order them
2. 3.8 bits/symbol
3. 3.8 bits/photon
4. 0.27 bits / time slot

[?]



| | Number of Photon | Number of Permutation | Number of Bits per Symbol | Number of Bits per Photon | Number of Bits per Time Slots |
|---|---|---|---|---|---|
| 0 | 1.0 | 14.0 | 3.807354922057604 | 3.807354922057604 | 0.272 |
| 1 | 2.0 | 14.0 | 3.807354922057604 | 1.903677461028802 | 0.272 |
| 2 | 3.0 | 14.0 | 3.807354922057604 | 1.2691183073525347 | 0.272 |
| 3 | 4.0 | 14.0 | 3.807354922057604 | 0.951838730514401 | 0.272 |
| 4 | 5.0 | 14.0 | 3.807354922057604 | 0.7614709844115208 | 0.272 |
| 5 | 6.0 | 14.0 | 3.807354922057604 | 0.6345591536762674 | 0.272 |
| 6 | 7.0 | 14.0 | 3.807354922057604 | 0.5439078460082292 | 0.272 |
| 7 | 8.0 | 14.0 | 3.807354922057604 | 0.4759193652572005 | 0.272 |
| 8 | 9.0 | 14.0 | 3.807354922057604 | 0.42303943578417824 | 0.272 |
| 9 | 10.0 | 14.0 | 3.807354922057604 | 0.3807354922057604 | 0.272 |
| 10 | 11.0 | 14.0 | 3.807354922057604 | 0.3461231747325095 | 0.272 |
| 11 | 12.0 | 14.0 | 3.807354922057604 | 0.3172795768381337 | 0.272 |
| 12 | 13.0 | 14.0 | 3.807354922057604 | 0.2928734555428926 | 0.272 |
| 13 | 14.0 | 14.0 | 3.807354922057604 | 0.2719539230041146 | 0.272 |
| 14 | 15.0 | 14.0 | 3.807354922057604 | 0.25382366147050694 | 0.272 |
| 15 | 16.0 | 14.0 | 3.807354922057604 | 0.23795968262860026 | 0.272 |
| 16 | 17.0 | 14.0 | 3.807354922057604 | 0.2239620542386826 | 0.272 |
| 17 | 18.0 | 14.0 | 3.807354922057604 | 0.21151971789208912 | 0.272 |
| 18 | 19.0 | 14.0 | 3.807354922057604 | 0.20038710116092653 | 0.272 |
| 19 | 20.0 | 14.0 | 3.807354922057604 | 0.1903677461028802 | 0.272 |

**Figure 4.1:** Table

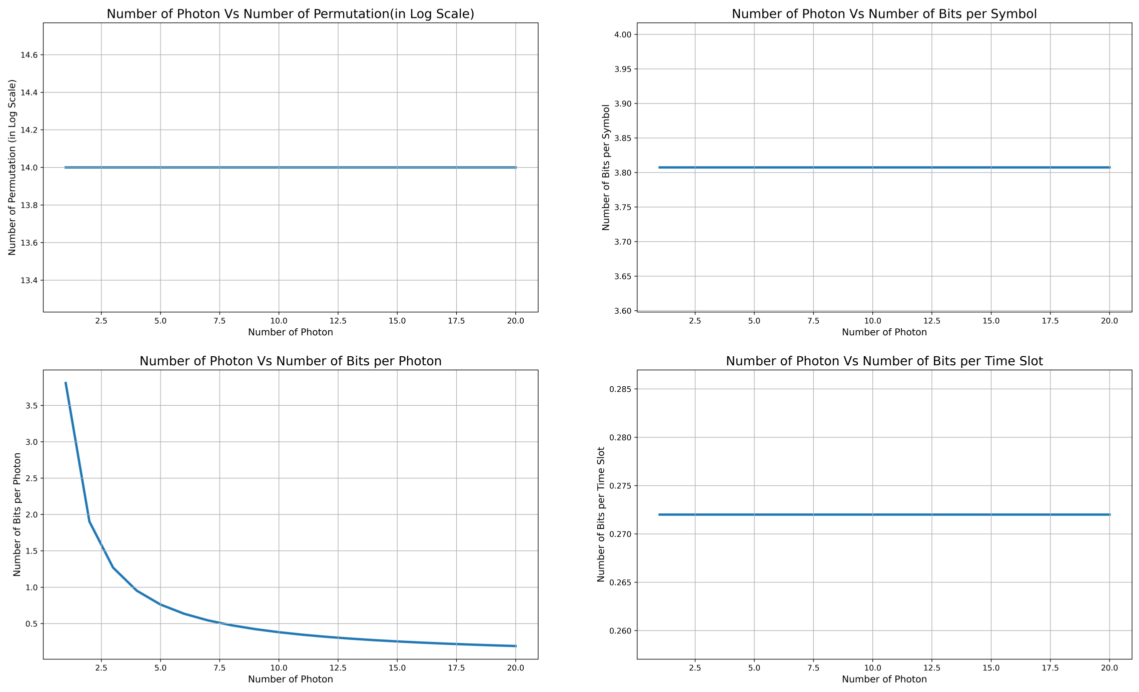
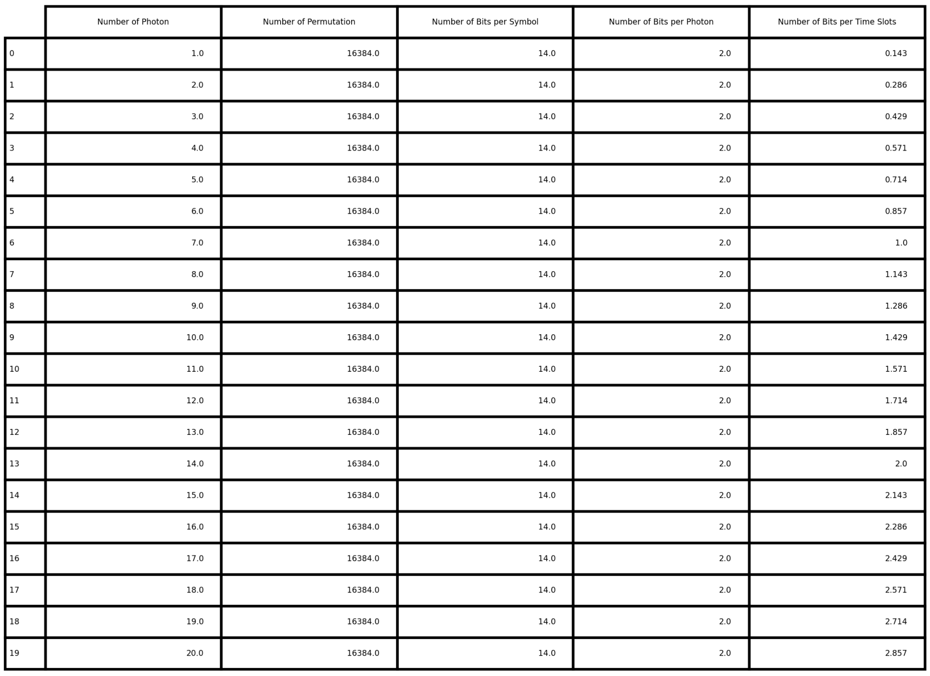


**Figure 4.2:** Number of Photons Vs Number of Permutation, Number of Photons Vs Number of Bits per Symbol, Number of Photons Vs Number of Bits per Photon, Number of Photons Vs Number of Bits per Time Slot

# 5

# On-Off Key (OOK)

OOK uses 7 photons in average in 14 time slots
It means that PPM has:
1. $2^{14} = 16,384$ ways to order them
2. 14 bits/symbol
3. 2 bits/photon
4. 1 bits / time slot



| | Number of Photon | Number of Permutation | Number of Bits per Symbol | Number of Bits per Photon | Number of Bits per Time Slots |
|---|---|---|---|---|---|
| 0 | 1.0 | 16384.0 | 14.0 | 2.0 | 0.143 |
| 1 | 2.0 | 16384.0 | 14.0 | 2.0 | 0.286 |
| 2 | 3.0 | 16384.0 | 14.0 | 2.0 | 0.429 |
| 3 | 4.0 | 16384.0 | 14.0 | 2.0 | 0.571 |
| 4 | 5.0 | 16384.0 | 14.0 | 2.0 | 0.714 |
| 5 | 6.0 | 16384.0 | 14.0 | 2.0 | 0.857 |
| 6 | 7.0 | 16384.0 | 14.0 | 2.0 | 1.0 |
| 7 | 8.0 | 16384.0 | 14.0 | 2.0 | 1.143 |
| 8 | 9.0 | 16384.0 | 14.0 | 2.0 | 1.286 |
| 9 | 10.0 | 16384.0 | 14.0 | 2.0 | 1.429 |
| 10 | 11.0 | 16384.0 | 14.0 | 2.0 | 1.571 |
| 11 | 12.0 | 16384.0 | 14.0 | 2.0 | 1.714 |
| 12 | 13.0 | 16384.0 | 14.0 | 2.0 | 1.857 |
| 13 | 14.0 | 16384.0 | 14.0 | 2.0 | 2.0 |
| 14 | 15.0 | 16384.0 | 14.0 | 2.0 | 2.143 |
| 15 | 16.0 | 16384.0 | 14.0 | 2.0 | 2.286 |
| 16 | 17.0 | 16384.0 | 14.0 | 2.0 | 2.429 |
| 17 | 18.0 | 16384.0 | 14.0 | 2.0 | 2.571 |
| 18 | 19.0 | 16384.0 | 14.0 | 2.0 | 2.714 |
| 19 | 20.0 | 16384.0 | 14.0 | 2.0 | 2.857 |

**Figure 5.1:** Table

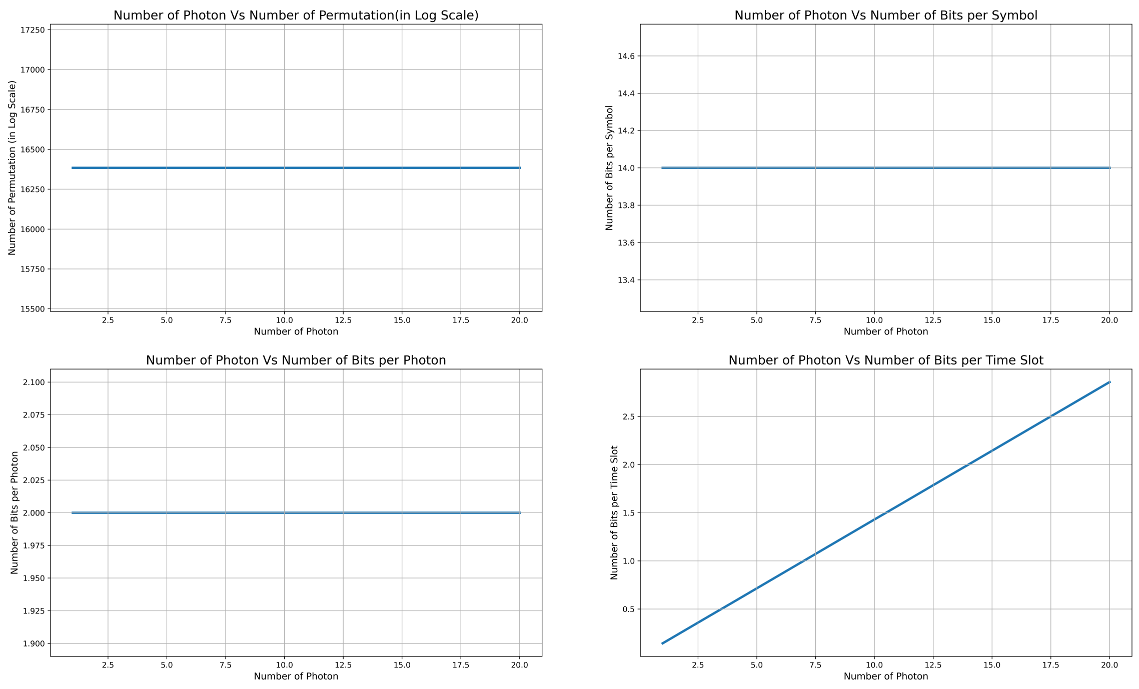


**Figure 5.2:** Number of Photons Vs Number of Permutation, Number of Photons Vs Number of Bits per Symbol, Number of Photons Vs Number of Bits per Photon, Number of Photons Vs Number of Bits per Time Slot

# 6

# In General

In general, it takes 4 photons in 14 time slots
It means that it has:
1. 1,001 ways to order them
2. 10 bits/symbol
3. 2.5 bits/photon
4. 0.71 bits / time slot

# 7 Galois Fields

If p is a prime number, it is possible to define a field with $p^m$ elements for any m. The fields, denoted $GF(p^m)$, are comprised of the polynomials of degree m-1 over the field $Z_p$. These polynomials are expressed as $a_{m-1}x^{m-1} + \cdots + a_1x^1 + a_0x^0$ where the coefficients $a_i$ take on values in the set $\{0, 1, \ldots, p-1\}$.

$$A_{4\times 3} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{bmatrix}$$

# 8 Reed-Solomon Codes

The Reed-Solomon (RS) codes are the non-binary codes, they are important for the use in communication systems where errors appear in bursts rather than independent random errors.

RS codes were discovered by Reed and Solomon in 1960. The encoding process assumes a code of RS(N,K) which results in N code words of length N symbols each storing K symbols of data, being generated, that are then sent over an erasure channel. The non-binary BCH block codes have $2^m(\{0, 1, 2, \ldots, 2^m - 1\})$ symbols with block length $n = 2^m - 1$, which can be extended to $n = 2^m$ or $m = 2^m + 1$. RS codes can correct up to $e_0$ errors within a block of n symbols by using $n - k = n - 2e_0 = 2^m - 1 - 2e_0$ parity symbols. Or it can locate and correct up to $\frac{t}{2}$ erroneous symbols at unknown locations. As an erasure code, it can correct up to t erasures at location that are known and provided to the algorithm, or it can detect and correct combinations of errors and erasures.

RS code can achieve the maximum number of error correction by finding the largest possible $d_{min} = 2e_0 + 1$

Any combination of K code words received at the other end is enough to reconstruct all of the N code words. The code rate is generally set to $\frac{1}{2}$ unless the channel's erasure likelihood can be adequately modelled and is seen to be less. In conclusion, N is usually 2K , meaning that at least half of all the code words sent must be received in order to reconstruct all of the code words sent.

We construct the encoding matrix E with dimension $k \times n$ where k is the number of check packets and n is the number of data packets.

$$E = \begin{bmatrix} 1 & 1 & 6 \\ 4 & 3 & 2 \\ 5 & 2 & 2 \\ 5 & 3 & 4 \\ 4 & 2 & 4 \end{bmatrix}$$

When the $n \times 1$ vector of data words is multiplied by E, an $k \times 1$ vector of check values is produced.

$$\begin{bmatrix} 1 & 1 & 6 \\ 4 & 3 & 2 \\ 5 & 2 & 2 \\ 5 & 3 & 4 \\ 4 & 2 & 4 \end{bmatrix} \times \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ 4 \\ 3 \\ 2 \end{bmatrix}$$

The values contained in the 8 'packets' that are sent: (0,4), (1,5), (2,6),(3,3),(4,5),(5,4),(6,3),(7,2). We observe that each of the 8 packets has an identifier that allows the recipient to determine exactly which packets of a Forward Error Correction (FEC) group have been received.

$$V = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 2 & 4 \\ 1 & 3 & 5 \\ 1 & 4 & 6 \\ 1 & 5 & 7 \\ 1 & 6 & 2 \\ 1 & 7 & 3 \end{bmatrix}$$

We have a transformed matrix D from Vandermonde matrix.

$$D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 6 \\ 4 & 3 & 2 \\ 5 & 2 & 2 \\ 5 & 3 & 4 \\ 4 & 2 & 4 \end{bmatrix}$$

For a collection of n packets including both data and check packets have been received, we extract n rows of the D matrix corresponding the n received packets. We call this $n \times n$ matrix $D^{'}$.

$$D^{'} = \begin{bmatrix} 1 & 1 & 6 \\ 4 & 3 & 2 \\ 5 & 2 & 2 \end{bmatrix}$$

Inverting $D^{'}$ yields $D^{'-1}$.

$$D^{'-1} = \begin{bmatrix} 5 & 5 & 2 \\ 5 & 7 & 3 \\ 3 & 3 & 3 \end{bmatrix}$$

The receiver knows the algorithms by which the check packets were constructed as follow:

$$\begin{bmatrix} 1 & 1 & 6 \\ 4 & 3 & 2 \\ 5 & 2 & 2 \end{bmatrix} \times \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ 4 \end{bmatrix}$$

Multiplying the received check value by $D'^{-1}$ recovers the original values.

$$\begin{bmatrix} 5 & 5 & 2 \\ 5 & 7 & 3 \\ 3 & 3 & 3 \end{bmatrix} \times \begin{bmatrix} 3 \\ 5 \\ 4 \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$