

Data Hiding Techniques Using Prime and Natural Numbers

Sandipan Dey,
Cognizant Technology Solutions,
Kolkata, India
sandipan.dey@gmail.com

Ajith Abraham,
Centre for Quantifiable Quality of Service in Communication Systems,
Norwegian University of Science and Technology
O.S. Bragstads plass 2E, N-7491 Trondheim, Norway
ajith.abraham@ieee.org

Bijoy Bandyopadhyay,
Department of Radio Physics and Electronics,
University of Calcutta
Kolkata, India
b_bandyopadhyay@yahoo.com

Sugata Sanyal,
School of Technology and Computer Science
Tata Institute of Fundamental Research,
Homi Bhabha Road, Mumbai - 400005, India
sanyal@tifr.res.in

Abstract

In this paper, a few novel data hiding techniques are proposed. These techniques are improvements over the classical LSB data hiding technique and the Fibonacci LSB data-hiding technique proposed by Battisti et al. [1]. The classical LSB technique is the simplest, but using this technique it is possible to embed only in first few bit-planes, since image quality becomes drastically distorted when embedding in higher bit-planes. Battisti et al. [1] proposed an improvement over this by using Fibonacci decomposition technique and generating a different set of virtual bit-planes all together, thereby increasing the number of bit-planes. In this paper, first we mathematically model and generalize this particular approach of virtual bit-plane generation. Then we propose two novel

embedding techniques, both of which are special-cases of our generalized model. The first embedding scheme is based on decomposition of a number (pixel-value) in sum of prime numbers, while the second one is based on decomposition in sum of natural numbers. Each of these particular representations generates a different set of (virtual) bit-planes altogether, suitable for embedding purposes. They not only allow one to embed secret message in higher bit-planes but also do it without much distortion, with a much better stego-image quality, in a reliable and secured manner, guaranteeing efficient retrieval of secret message. A comparative performance study between the classical Least Significant Bit (LSB) method, the data hiding technique using Fibonacci -p-Sequence decomposition and our proposed schemes has been done. Theoretical analysis indicates that image quality of the stego-image hidden by the technique using Fibonacci decomposition improves against simple LSB substitution method, while the same using the prime decomposition method improves drastically against that using Fibonacci decomposition technique, and finally the natural number decomposition method is a further improvement against that using prime decomposition technique. Also, optimality for the last technique is proved. For both of our data-hiding techniques, the experimental results show that, the stego-image is visually indistinguishable from the original cover image.

Keywords

Data hiding, Information Security, LSB, Fibonacci, Image Quality, Chebysev Inequality, Prime Number Theorem, Sieve of Eratosthenes, Goldbach Conjecture, Pigeon-hole Principle, Newton-Raphson method.

1 Introduction

Data hiding technique is a new kind of secret communication technology. It has been a hot research topic in recent years, and it is mainly used to convey messages secretly by concealing the presence of communication. While cryptography scrambles the message so that it cannot be understood, steganography hides the data so that it cannot be observed. The main objectives of the steganographic algorithms are to provide confidentiality, data integrity and authentication.

Most steganographic techniques proceed in such a way that the data which has to be hidden inside an image or any other medium like audio, video etc., is broken down into smaller pieces and they are inserted into appropriate locations in the medium in order to hide them. The aim is to make them un-perceivable and to leave no doubts in minds of the hackers who 'step into' media-files to uncover 'useful' information from them. To achieve this goal the critical data has to be hidden in such a way that there is no major difference between the original image and the 'corrupted' image. Only the authorized person knows about the presence of data. The algorithms can make use of the various properties of the image to embed the data without causing easily detectable changes in

them. Data embedding or water marking algorithms [3], [6], [7], [8], [14], [20] necessarily have to guarantee the following:

- Presence of embedded data is not visible.
- Ordinary users of the document/image are not affected by the watermark, i.e., a normal user does not see any ambiguity in the clarity of the document/image.
- The watermark can be made visible/retrievable by the creator (and possibly the authorized recipients) when needed; this implies that only the creator has the mechanism to capture the data embedded inside the document/image.
- The watermark is difficult for the other eavesdropper to comprehend and to extract them from the channels.

In this paper, we mainly discuss about using some new decomposition methods in a classical Image Domain Technique, namely LSB technique (Least Significant Bit coding, [18], [19], in order to make the technique more secure and hence less predictable. We basically generate an entirely new set of bit planes and embed data bit in these bit planes, using our novel decomposition techniques [40], [41].

For convenience of description, here, the LSB is called the 0^{th} bit, the second LSB is called the 1^{st} bit, and so on. We call the newly-generated set of bit-planes 'virtual', since we do not get these bit-planes in classical binary decomposition of pixels.

Rest of the paper is organized as follows: Sections 2 and 3 describe the embedding technique in classical LSB and Fibonacci decomposition technique with our modification. Section 4 describes a generalized approach that we follow in our novel data-hiding techniques using prime/natural number decomposition. Section 5 illustrates the embedding technique using the prime decomposition, while the experimental results obtained using this technique are reported in Section 6. In Section 7, we propose the natural number based embedding technique and the experimental results obtained are reported in Section 8. Finally, in Section 9 we draw our conclusions.

2 The Classical LSB Technique - Data Hiding by Simple LSB Substitution

Among many different data hiding techniques proposed to embed secret message within images, the LSB data hiding technique is one of the simplest methods for inserting data into digital signals in noise free environments, which merely embeds secret message-bits in a subset of the LSB planes of the image. Probability of changing an LSB in one pixel is not going to affect the probability of changing the LSB of the adjacent or any other pixel in the image. Data

hiding tools, such as Steganos, StegoDos, HideBSeek etc are based on the LSB replacement in the spatial domain [2]. But the LSB technique has the following major disadvantages:

- It is more predictable and hence less secure, since there is an obvious statistical difference between the modified and unmodified part of the stego-image.
- Also, as soon as we go from LSB to MSB for selection of bit-planes for our message embedding, the distortion in stego-image is likely to increase exponentially, so it becomes impossible (without noticeable distortion and with exponentially increasing distance from cover-image and stego-image) to use higher bit-planes for embedding without any further processing.

The workarounds may be: Through the random LSB replacement (in stead of sequential), secret messages can be randomly scattered in stego-images, so the security can be improved.

Also, using the approaches given by variable depth LSB algorithm [21], or by the optimal substitution process based on genetic algorithm and local pixel adjustment [4], one is able to hide data to some extent in higher bit-planes as well.

We propose two novel new data-hiding schemes by increasing the available number of bit-planes using new decomposition techniques. Similar approach was given using Fibonacci-p-sequence decomposition technique [1], [12], but we show the proposed decomposition techniques to be more efficient in terms of generating more virtual bit-planes and maintaining higher quality of stego-image after embedding.

3 Generalized Fibonacci LSB Data Hiding Technique

This particular technique, proposed by Battisti et al. [1], investigates a different bit-planes decomposition, based on the Fibonacci-p-sequences, given by,

$$F_p(0) = F_p(1) = \dots = F_p(p) = 1$$

$$F_p(n) = F_p(n-1) + F_p(n-p-1), \forall n \geq p+1, n, p \in \mathbb{N} \quad (1)$$

This technique basically uses Fibonacci-p-sequence decomposition, rather than classical binary decomposition (LSB technique) to obtain different set of bit-planes, embed a secret message-bit into a pixel if it passes the Zeckendorf condition, then while extraction, follow the reverse procedure.

We shall slightly modify the above technique, but before that let us first generalize our approach, put forward a mathematical model and then propose our new data-hiding techniques as special-cases of the generalized model.

For the proposed data hiding techniques our aim will be

- To expand the set of bit-planes and obtain a new different set of virtual bit-planes.
- To embed secret message in higher bit-planes of the cover-image as well, maintaining high image quality, i.e., without much distortion.
- To extract the secret message from the embedded cover-image efficiently and without error.

4 A Generalized LSB Data Hiding Technique

If we have k -bit cover image, there are only k available bit-planes where secret data can be embedded. Hence we try to find a function f that increases the number of bit-planes from k to n , $n \geq k$, by converting the k -bit 8-4-2-1 standard binary pixel representation to some other binary number system with different weights. We also have to ensure less distortion in stego-image with increasing bit plane. As is obvious, in case of classical binary decomposition, the mapping f is identity mapping. But, our job is to find a non-identity mapping that satisfies our end. Figure-1 presents our generalized model, while Figure-2 explains the process of embedding.

4.1 The Number System

We define a number system by defining the following:

- Base (radix) r (digits of the number system $\in \{0, \dots, r-1\}$)
- Weight function $W(\cdot)$, where $W(i)$ denotes the weight corresponding to i^{th} digit (e.g., for 8-4-2-1 binary system, $W(0) = 1$, $W(1) = 2$, $W(2) = 4$, $W(4) = 8$).

Hence, the pair $(r, W(\cdot))$, defines a number system completely. Obviously, our decimal system can be denoted in this notation as $(10, 10^{(\cdot)})$.

A number having representation $d_{k-1}d_{k-2} \dots d_1d_0$ in number system $(r, W(\cdot))$ will have the following value (in decimal), $D = \sum_{i=0}^{k-1} d_i \cdot W(i)$, $d_i \in \{0, 1, \dots, r-1\}$. This number system may have some redundancy if \exists more than one representation for the same value, e.g., the same (decimal) value D may be represented as $d_{k-1}d_{k-2} \dots d_1d_0$ and $d'_{k-1}d'_{k-2} \dots d'_1d'_0$, i.e., $D = \sum_{i=0}^{k-1} d_i \cdot W(i) = \sum_{i=0}^{k-1} d'_i \cdot W(i)$, where $d_i, d'_i \in \{0, 1, \dots, r-1\}$. Here $d_i \neq d'_i$ for at least 2 different i s.

To eliminate this redundancy and to ensure uniqueness, we should be able to represent one number uniquely in our number system. To achieve this, we must develop some technique, so that for number(s) having multiple (more than one, non-unique) representation in our number system, we can discard all representations but one. One way of doing this may be: from the multiple representations choose the one that has lexicographical highest (or lowest) value, discard all others. We shall use this shortly in case of our prime number system.

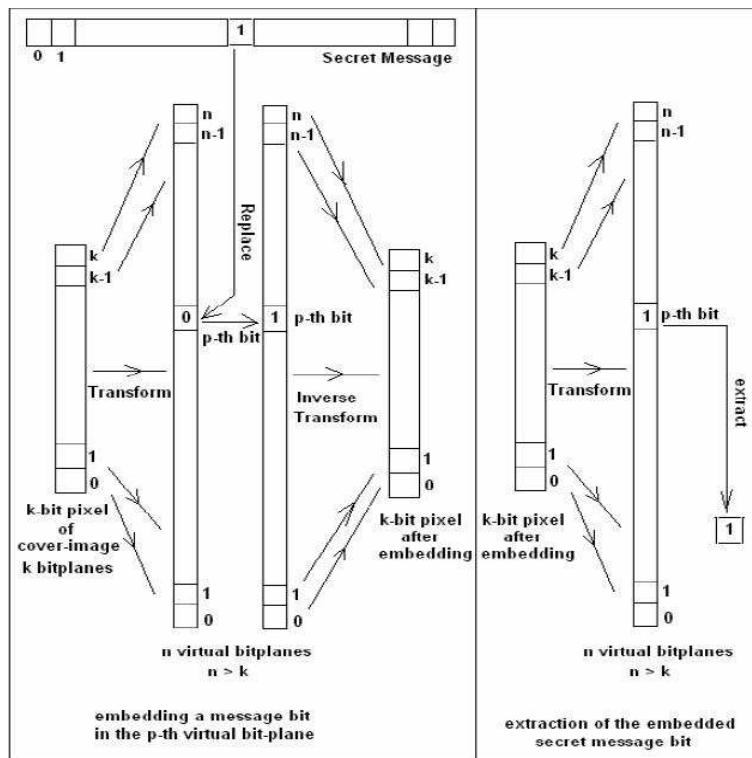


Figure 1: Basic block-diagram for generalized data-hiding technique

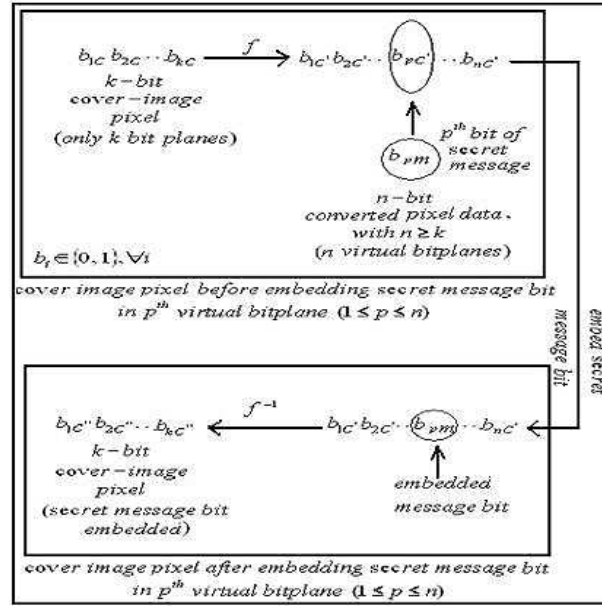


Figure 2: Basic block-diagram for embedding secret data-bit

As shown in Figure-2, for classical binary number system (8-4-2-1), we use the weight function $W(\cdot)$ defined by, $W(\cdot) = 2^{(\cdot)} \Rightarrow W : i \rightarrow 2^i \Rightarrow W(i) = 2^i, \forall i \in \mathbb{Z}^+ \cup \{0\}$, corresponding to i^{th} bit-plane (LSB = 0^{th} bit), so that a k-bit number (k-bit pixel-value) p_k is represented as $p_k = \sum_{i=0}^{k-1} b_{iC} \cdot 2^i$, where $b_{iC} \in \{0, 1\}$ - this is our well-known binary decomposition.

Now, our f converts this p_k to some virtual pixel representation p'_n (in a different binary number system) with n (virtual) bit-planes, obviously we need to have $n \geq k$ to expand number of bit planes. But finding such f is equivalent to finding a new weight function $W(\cdot)$, so that $W(i)$ denotes the weight of i^{th} (virtual) bit plane in our new binary number system, $\forall i \in \mathbb{Z}^+ \cup \{0\}$. Mathematically, $p'_n = \sum_{i=0}^{n-1} b'_{iC} \cdot W(i)$, where $b'_{iC} \in \{0, 1\}$ - this is our new decomposition, with the obvious condition that $(p_k)_{(2,2^{(\cdot)})} = (p'_n)_{(2,W(\cdot))}$.

Also, $W(i)$ must have less abrupt changes with respect to i , (i^{th} bit plane, virtual), than that in the case of 2^i , in order to have less distortion while embedding data in higher (virtual) bit planes. We call these expanded set of bit planes as virtual bit planes, since these were not available in the original cover image pixel data.

But, at the same time we must ensure the fact that the function f that we use must be injective, i.e., invertible, unless otherwise we shall not be able to extract the embedded message precisely.

4.2 The Number System Using Fibonacci p-Sequence Decomposition

Function f proposed by Battisti et al.[1] converts the pixel in binary decomposition to pixel in Fibonacci decomposition using generalized Fibonacci p-sequence, where corresponding weights are $F_p(n)$, $\forall n \in \mathbb{N}$, i.e., $W(.) = Fib_p(.)$, i.e., the number system proposed by them to model virtual bitplanes is $(2, F_p(.))$.

Since this number system too has redundancy (we can easily see it by applying pigeon-hole principle), for uniqueness and to make the transformation invertible, Zeckendorf's theorem, has been used.

4.2.1 Modification to ensure uniqueness

Instead of Zeckendorf's theorem, we use our lexicographically higher property. Hence, if a number has more than one representation using Fibonacci p-sequence decomposition, only the one lexicographically highest will be valid. Using this technique we prevent some redundancy also, since numbers in the range $[0, \sum_{i=0}^{n-1} F_p(i)]$ can be represented using n-bit Fibonacci-p-sequence decomposition. For an 8-bit image, the set of all possible pixel-values in the range $[0, 255]$ has the corresponding classical Fibonacci ($p = 1$, Fibonacci-1-sequence, Fibonacci series ([10], [11], [13])) decomposition as shown in Table-1. One may use this map to have a constant-time Fibonacci decomposition from pixel values into 12 virtual bit-planes.

5 The Prime Decomposition Technique

5.1 The Prime Number System and Prime Decomposition

We define a new number system, and as before we denote it as $(2, P(.))$, where the weight function $P(.)$ is defined as,

$$\begin{aligned}
 P(0) &= 1, \\
 P(i) &= p_i, \forall i \in \mathbb{Z}^+, \\
 p_i &= i^{th} \text{ Prime}, \\
 p_1 &= 2, p_2 = 3, p_3 = 5, \dots \\
 p_0 &= 1
 \end{aligned} \tag{2}$$

Since the weight function here is composed of prime numbers, we name this number system as prime number system and the decomposition as prime decomposition.

As we have discussed earlier, if a number has more than one representation in our number system, we always choose the lexicographically highest of them as valid, e.g., '3' has two different representations in 3-bit prime number system, namely, 100 and 011, since we have,

N	Fib Decomp	N	Fib Decomp	N	Fib Decomp	N	Fib Decomp
0	000000000000	64	000100010001	128	001010001000	192	010010100001
1	000000000001	65	000100010010	129	001010001001	193	010010100010
2	000000000010	66	000100010100	130	001010001010	194	010010100100
3	000000000100	67	000100010101	131	001010010000	195	010010100101
4	000000000101	68	000100100000	132	001010010001	196	010010101000
5	000000001000	69	000100100001	133	001010010010	197	010010101001
6	000000001001	70	000100100010	134	001010010100	198	010010101010
7	000000001010	71	000100100100	135	001010010101	199	010100000000
8	000000010000	72	000100100101	136	001010100000	200	010100000001
9	000000010001	73	000100101000	137	001010100001	201	010100000010
10	000000010010	74	000100101001	138	001010100010	202	010100000010
11	000000010100	75	000100101010	139	001010100100	203	010100000101
12	000000010101	76	000101000000	140	001010100101	204	010100001000
13	000000100000	77	000101000001	141	001010101000	205	010100001001
14	000000100001	78	000101000010	142	001010101001	206	010100001010
15	000000100010	79	000101000100	143	001010101010	207	010100001000
16	000000100100	80	000101000101	144	010000000000	208	010100010001
17	000000100101	81	000101001000	145	010000000001	209	010100010010
18	000000101000	82	000101001001	146	010000000010	210	010100010100
19	000000101001	83	000101001010	147	010000000100	211	010100010101
20	000000101010	84	000101010000	148	010000000101	212	010100100000
21	000001000000	85	000101010001	149	010000001000	213	010100100001
22	000001000001	86	000101010010	150	010000001001	214	010100100010
23	000001000010	87	000101010100	151	010000001010	215	010100100100
24	000001000100	88	000101010101	152	010000010000	216	010100100101
25	000001000101	89	001000000000	153	010000010001	217	010100010000
26	000001001000	90	001000000001	154	010000010010	218	010100010001
27	000001001001	91	001000000010	155	010000010100	219	010100010100
28	000001001010	92	001000000100	156	010000010101	220	010101000000
29	000001010000	93	001000000101	157	010000100000	221	010101000001
30	000001010001	94	001000001000	158	010000100001	222	010101000010
31	000001010010	95	001000001001	159	010000100010	223	010101000010
32	000001010100	96	001000001010	160	010000100100	224	010101000010
33	000001010101	97	001000010000	161	010000100101	225	010101001000
34	000010000000	98	001000010001	162	010000101000	226	010101001001
35	000010000001	99	001000010010	163	010000101001	227	010101001010
36	000010000010	100	001000010100	164	010000101010	228	010101010000
37	000010000100	101	001000010101	165	010001000000	229	010101010001
38	000010000101	102	001000100000	166	010001000001	230	010101010010
39	000010001000	103	001000100001	167	010001000010	231	010101010100
40	000010001001	104	001000100010	168	010001000010	232	010101010101
41	000010001010	105	001000100100	169	010001000101	233	100000000000
42	000010010000	106	001000100101	170	010001001000	234	100000000001
43	000010010001	107	001000101000	171	010001001001	235	100000000010
44	000010010010	108	001000101001	172	010001001010	236	100000000010
45	000010010100	109	001000101010	173	010001010000	237	100000000010
46	000010010101	110	001001000000	174	010001010001	238	100000001000
47	000010100000	111	001001000001	175	010001010010	239	100000001001
48	000010100001	112	001001000010	176	010001010100	240	100000001010
49	000010100010	113	001001000100	177	010001010101	241	100000010000
50	000010100100	114	001001000101	178	010010000000	242	100000010001
51	000010100101	115	001001001000	179	010010000001	243	100000010010
52	000010101000	116	001001001001	180	010010000010	244	100000010100
53	000010101001	117	001001001010	181	010010000100	245	100000010101
54	000010101010	118	001001010000	182	010010000101	246	100000100000
55	000100000000	119	001001010001	183	010010001000	247	100000100001
56	000100000001	120	001001010010	184	010010001001	248	100000100010
57	000100000010	121	001001010100	185	010010001010	249	100000100100
58	000100000010	122	001001010101	186	010010010000	250	100000100101
59	000100000101	123	001010000000	187	010010010001	251	100000101000
60	000100001000	124	001010000001	188	010010010010	252	100000101001
61	000100001001	125	001010000010	189	010010010100	253	100000101010
62	000100001010	126	001010000100	190	010010010101	254	100001000000
63	000100010000	127	001010000101	191	010010100000	255	100001000001

Table 1: Fibonacci (1-sequence) decomposition for 8-bit image yielding 12 virtual bit-planes

$$\begin{aligned}
1.P(2) + 0.P(1) + 0.P(0) &= 1.p2 + 0.p1 + 0.1 = 1.3 + 0.2 + 0.1 = 3 \\
0.P(2) + 1.P(1) + 1.P(0) &= 0.p2 + 1.p1 + 1.1 = 0.3 + 1.2 + 1.1 = 3
\end{aligned}$$

100 being lexicographically (from left to right) higher than 011, we choose 100 to be valid representation for 3 in our prime number system and hence discard 011, which is no longer a valid representation in our number system.

$$3 \equiv \max_{\text{lexicographic}}(100, 011) \equiv 100.$$

Hence, for our 3-bit example, the valid representations are: 000 \leftrightarrow 0, 001 \leftrightarrow 1, 010 \leftrightarrow 2, 100 \leftrightarrow 3, 101 \leftrightarrow 4, 110 \leftrightarrow 5, 111 \leftrightarrow 6. Numbers in the range [0, 6] can be decomposed using our 3-bit prime number system uniquely, with only the representation 011 avoided.

Now, let us proceed with this very simplified example to see how the secret data bit is going to be embedded. We shall embed a secret data bit into a (virtual) bit-plane by just simply replacing the corresponding bit by our data bit, if we find that after embedding, the resulting representation is a valid representation in our number system, otherwise we do not embed, just skip. This is only to guarantee the existence of the inverse function and proper extraction of our secret embedded message bit.

Again, let us elucidate by our previous 3-bit example. Let the 3-bit pixel within which we want to embed secret data be of value 2, use prime decomposition to get 010, and we want to embed in the LSB bit-plane, let our secret message bit to be embedded be 1. So, we just replace the pixel LSB 0 by data bit 1 and immediately see that after embedding the pixel, it will become 011, which is not a valid representation, hence we skip this pixel without embedding our secret data bit.

Had we used this pixel value for embedding and after embedding ended up with pixel value 011 (value 3), we might get erroneous result while extraction of the secret bit. Because during extraction decomposition of embedded pixel value 3 would wrongly give 100 instead of 011, and extraction of LSB virtual bit-plane would wrongly give the embedded bit as 0 instead of its true value 1. Figure-3 explains this error pictorially.

Hence, embed secret data bit **only** to those pixels, where after embedding, we get a valid representation in the number system.

5.2 Embedding algorithm

- First we find the set of all prime numbers that are required to decompose a pixel value in a k-bit cover-image, i.e., we need to find a number $n \in \mathbb{N}$ such that all possible pixel values in the range $[0, 2^k - 1]$ can be represented using first n primes in our n-bit prime number system, so that we get n virtual bit-planes after decomposition. We can use Sieve method, for example, to find primes. (To find the n is quite easy, since we see, using Goldbach conjecture etc, that all pixel-values in the range $[0, \sum_{i=0}^{m-1} p_i]$

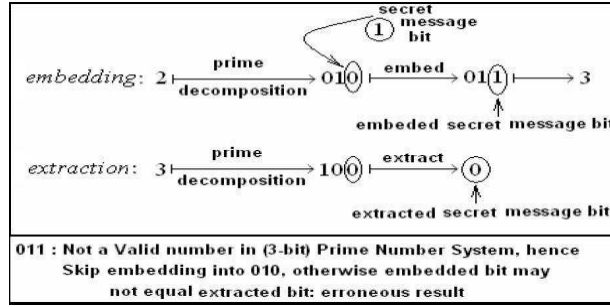


Figure 3: Error in not guaranteeing uniqueness of transformation

can be represented in our m -bit prime number system, so all we need to do is to find an n such that $\sum_{i=0}^{n-1} p_i \geq 2^k - 1$, since the highest number that can be represented in n -bit prime number system is $\sum_{i=0}^{n-1} p_i$.

- After finding the primes, we create a map of k -bit (classical binary decomposition) to n -bit numbers (prime decomposition), $n > k$, marking all the valid representations (as discussed in previous section) in our prime number system. For an 8-bit image the set of all possible pixel-values in the range $[0, 255]$ has the corresponding prime decomposition as shown in Table-2. As one may notice, the size of the map to be stored has been increased in this case, indicating a slightly greater space complexity.
- Next, for each pixel of the cover image, we choose a (virtual) bit plane, say p^{th} bit-plane and embed the secret data bit into that particular bit plane, by replacing the corresponding bit by the data bit, if and only if we find that after embedding the data bit, the resulting sequence is a valid representation in n -bit prime number system, i.e., exists in the map otherwise discard that particular pixel for data hiding.
- After embedding the secret message bit, we convert the resultant sequence in prime number system back to its value (in classical 8-4-2-1 binary number system) and we get our stego-image. This reverse conversion is easy, since we need to calculate $\sum_{i=0}^{n-1} b_i \cdot p_i$ only, where $b_i \in \{0, 1\}, \forall i \in \{0, n-1\}$

5.3 Extraction algorithm

The extraction algorithm is exactly the reverse. From the stego-image, we convert each pixel with embedded data bit to its corresponding prime decomposition and from the p^{th} bit-plane extract the secret message bit. Combine all the bits to get the secret message. Since, for efficient implementation, we shall have a hash-map for this conversion, the bit extraction is constant-time, so the secret message extraction will be polynomial (linear) in the length of the message embedded.

N	Prime Decomp	N	Prime Decomp	N	Prime Decomp	N	Prime Decomp
0	000000000000000	64	100000100000010	128	111000000010000	192	111110000100000
1	000000000000001	65	1000001000000100	129	111000000010001	193	111110000100001
2	000000000000010	66	100001000000000	130	111000000010010	194	111110000100000
3	000000000000100	67	100001000000001	131	111000000010100	195	111110000100001
4	000000000000101	68	100001000000010	132	111000000100000	196	111110000100010
5	0000000000001000	69	1000010000000100	133	111000000100001	197	1111100001000100
6	0000000000001001	70	1000010000000101	134	111000000100000	198	111110010000000
7	00000000000010000	71	1000010000001000	135	111000000100001	199	111110010000001
8	00000000000010001	72	100010000000000	136	1110000001000010	200	111110100000000
9	00000000000010010	73	100010000000001	137	1110000001000100	201	111110100000001
10	00000000000010100	74	100100000000000	138	111000010000000	202	111110100000010
11	000000000000100000	75	100100000000001	139	111000010000001	203	1111101000000100
12	0000000000000001	76	1001000000000010	140	111000010000000	204	111111000000000
13	0000000000000000	77	1001000000000100	141	111000010000001	205	111111000000001
14	0000000000000001	78	1001000000000101	142	111000010000010	206	111111000000010
15	00000000000000010	79	10010000000001000	143	11100001000001000	207	1111110000000100
16	0000000000000100	80	101000000000000	144	111000100000000	208	1111110000000101
17	0000000000000000	81	101000000000001	145	111000100000001	209	1111110000000100
18	00000000000000001	82	1010000000000010	146	1110001000000010	210	11111100000001001
19	00000000000000000	83	10100000000000100	147	11100010000000100	211	1111110000000000
20	00000000000000001	84	110000000000000	148	1110001000000101	212	1111110000000001
21	00000000000000010	85	110000000000001	149	11100010000001000	213	11111100000001010
22	0000000000000100	86	1100000000000010	150	111000000000000	214	111111000000010100
23	0000000000000000	87	1100000000000100	151	111000000000001	215	1111110000000000
24	00000000000000001	88	1100000000000101	152	111000000000000	216	1111110000000001
25	0000000000000010	89	11000000000001000	153	111000000000001	217	1111110000000000
26	00000000000000100	90	11000000000001001	154	111000000000010	218	1111110000000001
27	0000000000000101	91	11000000000001000	155	1110000000000100	219	1111110000000010
28	0000000000000000	92	11000000000001001	156	1110000000000101	220	1111110000000100
29	00000000000000001	93	110000000000010010	157	11100000000001000	221	111111010000000
30	00000000000000000	94	11000000000001000	158	11100000000001001	222	111111010000001
31	00000000000000001	95	110000000000000	159	111000000000000	223	111111000000000
32	00000000000000001	96	110000000000001	160	111000000000001	224	1111110000000001
33	00000000000000010	97	110000000000000	161	111000000000010	225	1111110000000010
34	000000000000000100	98	110000000000001	162	11100000000001010	226	1111110000000100
35	000000000000000100	99	110000000000010	163	111000000000000	227	1111110000000101
36	0000000000000000	100	1100000000000100	164	111000000000001	228	11111100000001000
37	00000000000000001	101	110000000000000	165	111000000000000	229	11111100000001001
38	000000000000000010	102	110000000000001	166	111000000000001	230	1111110000000000
39	00000000000000010	103	110000000000000	167	111000000000010	231	1111110000000001
40	000000000000000100	104	1100000000000001	168	1110000000000100	232	1111110000000010
41	0000000000000000	105	1100000000000010	169	111000000000000	233	11111100000001000
42	00000000000000001	106	1100000000000100	170	111000000000001	234	1111110000000000
43	00000000000000000	107	110000000000000	171	111000000000000	235	1111110000000001
44	00000000000000001	108	110000000000001	172	111000000000001	236	1111110000000000
45	00000000000000010	109	1100000000000010	173	1110000000000010	237	1111110000000001
46	000000000000000100	110	11000000000000100	174	11100000000000100	238	11111100000000010
47	000000000000000101	111	11000000000000101	175	111000000000000	239	11111100000000100
48	0000000000000001000	112	110000000000001000	176	1110000000000001	240	1111110000000000
49	0000000000000001001	113	110000000000000	177	1110000000000010	241	1111110000000001
50	0000000000000000	114	110000000000001	178	1110000000000100	242	1111110000000010
51	00000000000000001	115	110000000000000	179	1110000000000101	243	11111100000000100
52	00000000000000010	116	110000000000001	180	11100000000001000	244	11111100000000101
53	000000000000000100	117	1100000000000010	181	111000000000000	245	1111110000000000
54	0000000000000000	118	1100000000000010	182	111000000000001	246	1111110000000001
55	00000000000000001	119	1100000000000101	183	1110000000000010	247	1111110000000000
56	00000000000000000	120	11000000000001000	184	1110000000000100	248	1111110000000001
57	00000000000000001	121	110000000000000	185	1110000000000101	249	1111110000000010
58	00000000000000010	122	1100000000000001	186	11100000000001000	250	11111100000000100
59	000000000000000100	123	1100000000000010	187	11100000000001001	251	1111110000000000
60	0000000000000000	124	11000000000000100	188	111000000000000	252	1111110000000001
61	00000000000000001	125	11000000000000101	189	111000000000001	253	1111110000000000
62	00000000000000000	126	110000000000001000	190	111000000000010010	254	11111100000000001
63	00000000000000001	127	110000000000001001	191	111000000000010100	255	111111000000000010

Table 2: Prime decomposition for 8-bit image yielding 15 virtual bit-planes

5.4 The performance analysis : Comparison between classical Binary, Fibonacci and Prime Decomposition

In this section, we do a comparative study between the different decompositions and its effect upon higher-bit-plane data-hiding. We basically try to prove our following two claims, by means of the following theorems from Number Theory [39]:

5.4.1 The Prime Number Theorem : A Polynomial tight bound for Primes

By Tchebychef theorem, $0.92 < \frac{\pi(x) \ln(x)}{x} < 1.105$, $\forall x \geq 2$, where $\pi(x)$ denotes number of primes not exceeding x , i.e., $\pi(x) = \theta\left(\frac{x}{\ln x}\right)$. This leads to famous Prime Number theorem $\lim_{n \rightarrow \infty} \left(\frac{\pi(n)}{(n/\ln(n))}\right) = 1$. From this one can show [1] that, if p_n be the n^{th} prime, $\exists L1, L2 \in \mathfrak{R}$, such that $L1 < \left(\frac{p_n}{(n \ln(n))}\right) < L2$, $\forall n \geq 2$, $n \in \mathbb{Z}^+$, i.e., $\lim_{n \rightarrow \infty} \left(\frac{p_n}{(n \ln(n))}\right) = 1$.

$$p_n = \theta(n \cdot \ln(n)) \quad (3)$$

5.4.2 A lower bound for the Fibonacci-p-Sequence

The Fibonacci-p-sequence, for $p \geq 1$, $p \in \mathbb{N}$, is given by,

$$\begin{aligned} F_p(0) &= F_p(1) = \dots = F_p(p) = 1, \\ F_p(n) &= F_p(n-1) + F_p(n-p-1), \quad \forall n \geq p+1, n \in \mathbb{N} \end{aligned}$$

We prove the following lemmas and find

Lemma-1: If the ratio of two consecutive numbers in Fibonacci p-sequence converges to limit $\alpha_p \in \mathfrak{R}^+$, α_p satisfies the equation $x^{p+1} - x^p - 1 = 0$, $\forall p \in \mathbb{N}$.

Proof:

$$\begin{aligned} \alpha_p &= \lim_{n \rightarrow \infty} \left(\frac{f_{n+p}}{f_{n+p-1}}\right) = \lim_{n \rightarrow \infty} \left(\frac{f_{n+p-1}}{f_n}\right) = \dots = \lim_{n \rightarrow \infty} \left(\frac{f_n}{f_{n-1}}\right) = \dots, \\ f_n &= n^{\text{th}} \text{ number in the Fibonacci - } p \text{ Sequence, } f_{n+p} = f_{n+p-1} + f_{n-1} \\ &\Rightarrow \alpha_p = \lim_{n \rightarrow \infty} \left(\frac{f_{n+p-1} + f_{n-1}}{f_{n+p-1}}\right) = \lim_{n \rightarrow \infty} \left(\frac{f_n}{f_{n-1}}\right), \\ &\Rightarrow \alpha_p = 1 + \lim_{n \rightarrow \infty} \prod_{k=n-1}^{k=n+p-2} \left(\frac{f_k}{f_{k+1}}\right) = \lim_{n \rightarrow \infty} \left(\frac{f_n}{f_{n-1}}\right) \\ &\Rightarrow \alpha_p = 1 + \prod_{k=1}^{k=p} \left(\frac{1}{\alpha_p}\right) \Rightarrow \alpha_p = 1 + \frac{1}{\alpha_p^p} \end{aligned}$$

$$\Rightarrow \alpha_p^{p+1} - \alpha_p^p - 1 = 0$$

Lemma-2: If α_p be a +ve root of the equation $x^{p+1} - x^p - 1 = 0$, we have $1 < \alpha_p < 2, \forall p \in \mathbb{N}$.

Proof: We have,

$$\begin{aligned} \alpha_p^{p+1} - \alpha_p^p - 1 = 0 \text{ also, } 2^{p+1} - 2^p - 1 = 2^p - 1 > 0, \forall p \in \mathbb{Z}^+ \\ \Rightarrow 2^p - 1 > \alpha_p^{p+1} - \alpha_p^p - 1 \Rightarrow (2^p - \alpha_p^p) > \alpha_p^p(\alpha_p - 2) \end{aligned} \quad (4)$$

Also,

$$-1 < 0 = \alpha_p^{p+1} - \alpha_p^p - 1 \Rightarrow \alpha_p^p(\alpha_p - 1) > 0 \Rightarrow \alpha_p > 1 \text{ (since positive)} \quad (5)$$

From (4), we immediately see the following:

- $\alpha_p > 0$ according to our assumption, hence we can not have $\alpha_p = 2$ (LHS & RHS both becomes 0, that does not satisfy inequality (4)).
- If $\alpha_p > 2$, we have LHS < 0 while RHS > 0 which again does not satisfy inequality (4).
- Hence we have $\alpha_p < 2, \forall p \in \mathbb{N}$

From (5), we have, $\alpha_p > 1$. Combining, we get, $1 < \alpha_p < 2, \forall p \in \mathbb{N}$

Lemma-3: If α_p be a +ve root of the equation $x^{p+1} - x^p - 1 = 0$, where $p \in \mathbb{N}$, we have,

- $\alpha_k > \alpha_{k+1}$
- $\alpha_{k+1} > \frac{1+\alpha_k}{2}$
- $\alpha_k^k < (k+1), \forall k \in \mathbb{N}$

Proof: We have,

$$\begin{aligned} \text{For } p = k, \alpha_k^{k+1} - \alpha_k^k - 1 = 0 \\ \text{For } p = k+1, \alpha_{k+1}^{k+2} - \alpha_{k+1}^{k+1} - 1 = 0 \\ \Rightarrow \alpha_{k+1}^{k+1}(\alpha_{k+1} - 1) = \alpha_k^k(\alpha_k - 1) \\ \Rightarrow \left(\frac{\alpha_k}{\alpha_{k+1}} \right)^k = \left(\frac{\alpha_{k+1} - 1}{\alpha_k - 1} \right) \cdot \alpha_{k+1} \end{aligned} \quad (6)$$

From (6) we can argue,

- $\alpha_k \neq \alpha_{k+1}$, since neither of them is 0 or 1 (from lemma-2).

- If $\alpha_k < \alpha_{k+1}$, we have LHS of inequality (6) < 1 , but RHS > 1 , since both the terms in RHS will be greater than 1 (by our assumption and by lemma-2), a contradiction.
- Hence, we must have

$$\alpha_k > \alpha_{k+1}, \forall k \in \mathbb{N} \quad (7)$$

Again, from (6) we have,

$$\begin{aligned} \Rightarrow \left(\frac{\alpha_{k+1} - 1}{\alpha_k - 1} \right) \cdot \alpha_{k+1} &> 1, \text{ since } \left(\frac{\alpha_k}{\alpha_{k+1}} \right)^k > 1, \text{ from (7)} \\ \Rightarrow 2 > \alpha_{k+1} &> \left(\frac{\alpha_k - 1}{\alpha_{k+1} - 1} \right), \text{ (from lemma-2)} \\ &\Rightarrow \alpha_{k+1} > \frac{1 + \alpha_k}{2} \end{aligned} \quad (8)$$

Now, let us induct on p to prove $\alpha_p^p < p + 1$.

Base case: for $p = 1$, $\alpha_1 < 2$, by lemma-2

Let us assume the inequality holds $\forall p \leq k \Rightarrow \alpha_p^p < p + 1 \forall p \leq k$

Induction Step: for $p = k + 1$, $\alpha_{k+1}^{k+1} = \alpha_k^k \cdot \left(\frac{\alpha_k - 1}{\alpha_{k+1} - 1} \right)$, by (6)

$$\begin{aligned} \Rightarrow \alpha_{k+1}^{k+1} &< (k + 1) \cdot \left(\frac{\alpha_k - 1}{\alpha_{k+1} - 1} \right), \text{ by induction hypothesis} \\ &\Rightarrow \alpha_{k+1}^{k+1} < (k + 1) \cdot \left(1 + \frac{\alpha_k - \alpha_{k+1}}{\alpha_{k+1} - 1} \right) \\ &\Rightarrow \alpha_{k+1}^{k+1} < (k + 1) + \left(\frac{\alpha_k - \alpha_{k+1}}{\alpha_{k+1} - 1} \right) \\ \Rightarrow \alpha_{k+1}^{k+1} &< (k + 1) + 1, \left(\text{from (8), we have, } \frac{\alpha_k - \alpha_{k+1}}{\alpha_{k+1} - 1} < 1 \right) \\ &\Rightarrow \alpha_{k+1}^{k+1} < (k + 2) \\ &\Rightarrow \alpha_p^p < (p + 1), \forall p \in \mathbb{N} \end{aligned} \quad (9)$$

Lemma-4 The following inequalities always hold:

- $(k + 1)^{\frac{1}{k}} < k^{\frac{1}{k-1}} < \dots < 4^{\frac{1}{3}} < 3^{\frac{1}{2}} < 2$
- $\alpha_p^p < p + 1 \Rightarrow \alpha_p^{p-1} < p \Rightarrow \dots \alpha_p^3 < 4 \Rightarrow \alpha_p^2 < 3 \Rightarrow \alpha_p < 2$

Proof: By Binomial Theorem, we have,

$$\begin{aligned} (k + 1)^{k-1} &= \sum_{n=0}^{k-1} \frac{(k-1)!}{n!(k-1-n)!} \cdot k^n = 1 + \sum_{n=1}^{k-1} \frac{1}{n!} \cdot \prod_{r=1}^n \left(1 - \frac{r}{k}\right) \cdot k^{k-1} \\ &< \underbrace{(1 + 1 + 1 + \dots + 1)}_{k \text{ times}} \cdot k^{k-1} = k \cdot k^{k-1} = k^k \Rightarrow (k + 1)^{\frac{1}{k}} < k^{\frac{1}{k-1}} \end{aligned} \quad (10)$$

Hence, we have, $(k+1)^{\frac{1}{k}} < k^{\frac{1}{k-1}} < \dots < 4^{\frac{1}{3}} < 3^{\frac{1}{2}} < 2$

Also, from (9) we have, $\alpha_k < (k+1)^{\frac{1}{k}}$.

Combining, we get,

$$\alpha_k < (k+1)^{\frac{1}{k}} < k^{\frac{1}{k-1}} < \dots < 4^{\frac{1}{3}} < 3^{\frac{1}{2}} < 2$$

$$\alpha_k^k < (k+1) \Rightarrow \alpha_k^{k-1} < k \dots \Rightarrow \alpha_k^4 < 5 \Rightarrow \alpha_k^3 < 4 \Rightarrow \alpha_k^2 < 3 \Rightarrow \alpha_k < 2 \quad (11)$$

Lemma-5 The following inequality gives us the lower bound,

$$F_p(n) > \alpha_p^{n-p}, \quad \forall n > p, \quad n \in \mathbb{N} \quad (12)$$

where α_p is the +ve root of the equation $x^{p+1} - x^p - 1 = 0$.

Proof: We induct on n to show the result.

$F_p(0) = F_p(1) = \dots = F_p(p) = 1$, (By definition of Fibonacci-p-Sequence).

Base case :

$$n = p + 1, \quad F_p(p + 1) = F_p(p) + F_p(0) = 1 + 1 = 2 > \alpha_p, \quad (\text{From Lemma-4})$$

$$n = p + 2, \quad F_p(p + 2) = F_p(p + 1) + F_p(1) = 2 + 1 = 3 > \alpha_p^2, \quad (\text{From Lemma-4})$$

$$n = p + 3, \quad F_p(p + 3) = F_p(p + 2) + F_p(2) = 3 + 1 = 4 > \alpha_p^3, \quad (\text{From Lemma-4})$$

...

$$n = p + (p + 1), \quad F_p(p + p + 1) = F_p(p + p) + F_p(p) = (p + 1) + 1$$

$$= p + 2 > \alpha_p^{p+1}, \quad (\text{From Lemma-4})$$

Induction Step:

Let's assume the above result is true $\forall m < n$, $m, n \in \mathbb{N}$, for $m > 2p + 1$ as well. Then we have,

$$F_p(n) = F_p(n-1) + F_p(n-p-1) > \alpha_p^{n-p-1} + \alpha_p^{n-2p-1} \quad (\text{hypothesis})$$

$$\Rightarrow F_p(n) > \alpha_p^{n-2p-1} \cdot (1 + \alpha_p^p) = \alpha_p^{n-2p-1} \cdot \alpha_p^{p+1} = \alpha_p^{n-p}$$

$$\Rightarrow F_p(n) > \alpha_p^{n-p}, \quad \forall n > p, \quad n \in \mathbb{N}$$

Hence, we have the following inequality,

$$F_p(n) > (\alpha_p)^{n-p},$$

$$\alpha_p \in \mathbb{R}^+,$$

$$\alpha_1 = \frac{1 + \sqrt{5}}{2} \approx 1.618034,$$

$$\alpha_2 \approx 1.465575,$$

$$\alpha_3 \approx 1.380278,$$

$$\alpha_4 \approx 1.324718,$$

$$\alpha_p > \alpha_{p+1}, \quad \forall p \in \mathbb{Z}^+$$

$Fib_1(n)$	α_1^{n-1}	$Fib_1(n)$	α_1^{n-1}
2	1.618	3	2.618
5	4.236	8	6.854
13	11.090	21	17.944
34	29.034	55	46.979
89	76.013	144	122.992
233	199.006	377	321.998
610	521.004	987	843.002
1597	1364.007	2584	2207.010
4181	3571.018	6765	5778.029
10946	9349.051	17711	15127.086
28657	24476.146	46368	39603.247
75025	64079.418	121393	103682.706
196418	167762.190	317811	271445.002
514229	439207.365	832040	710652.646
1346269	1149860.461	2178309	1860513.836
3524578	3010375.477	5702887	4870891.223
9227465	7881269.791	14930352	12752166.014
24157817	20633443.895	39088169	33385622.999
63245986	54019088.074	102334155	87404745.343
165580141	141423888.869	267914296	228828723.934
433494437	370252757.977	701408733	599081716.807
1134903170	969334854.855	1836311903	1568417186.629
2971215073	2537753036.521	4807526976	4106171833.156
7778742049	6643927474.721	12586269025	10750103522.928
20365011074	17394037817.746	32951280099	28144152375.826
53316291173	45538208048.829	86267571272	73682389315.076
139583862445	119220644109.601	225851433717	192903109060.823
365435296162	312123875552.315	591286729879	505027182631.253
956722026041	817151378583.699	1548008755920	1322179079633.401
2504730781961	2139331297036.010	4052739537881	3461511733907.302
6557470319842	5600845227000.975	10610209857723	9062360514205.225
17167680177565	14663211490563.064	27777890035288	23725581307425.750

Table 3: α_1 is a +ve Root of $x^2 - x - 1 = 0$, i.e., $\alpha_1 \approx 1.618034$

The sequence α_p is decreasing in p.

The empirical results illustrated in Tables 3 and 4 also depict the same:

5.4.3 Measures

As we know, Security, embedding distortion and embedding rate can be used as schemes to evaluate the performance of the data hiding schemes. The following are the popular parameters,

- Entropy - A steganographic system is perfectly secure when the statistics of the cover-data and stego-data are identical, which means that the relative entropy between the cover data and the stego-data is zero. Entropy considers the information to be modeled as a probabilistic process that can be measured in a manner that agrees with intuition [38]. The information theoretic approach to steganography holds capacity of the system to be modeled as the ability to transfer information ([22], [23], [37]).
- Mean Squared Error and SNR - The (weighted) mean squared error between the cover image and the stego-image (embedding distortion) can be used as one of the measures to assess the relative perceptibility of the embedded text. Imperceptibility takes advantage of human psycho visual redundancy, which is very difficult to quantify. Mean square error (MSE) and Peak Signal to Noise Ratio (PSNR) can also be used as metrics to

$Fib_2(n)$	α_2^{n-2}	$Fib_2(n)$	α_2^{n-2}
2	1.466	3	2.148
4	3.148	6	4.613
9	6.761	13	9.909
19	14.523	28	21.284
41	31.193	60	45.716
88	67.000	129	98.194
189	143.910	277	210.910
406	309.104	595	453.013
872	663.923	1278	973.027
1873	1426.040	2745	2089.963
4023	3062.990	5896	4489.030
8641	6578.993	12664	9641.983
18560	14131.013	27201	20710.006
39865	30351.989	58425	44483.001
85626	65193.007	125491	95544.996
183916	140027.997	269542	205221.004
395033	300766.000	578949	440793.997
848491	646015.002	1243524	946781.002
1822473	1387574.999	2670964	2033590.001
3914488	2980371.002	5736961	4367946.001
8407925	6401536.002	12322413	9381907.004
18059374	13749853.006	26467299	20151389.008
38789712	29533296.012	56849086	43283149.019
83316385	63434538.027	122106097	92967834.041
178955183	136250983.061	262271568	199685521.092
384377665	292653355.137	563332848	428904338.205

Table 4: α_2 is a +ve Root of $x^3 - x^2 - 1 = 0$, i.e., $\alpha_2 \approx 1.465571$

measure the degree of imperceptibility:

$$MSE = \sum_{i=1}^M \sum_{j=1}^N (f_{ij} - g_{ij})^2 MN$$

$$PSNR = 10 \cdot \log_{10} \left(\frac{L^2}{MSE} \right)$$

where M and N are the number of rows and number of columns respectively of the cover image, f_{ij} is the pixel value from the cover image, g_{ij} is the pixel value from the stego-image, and L is the peak signal value of the cover image (for 8-bit images, $L = 255$. In general, for k -bit grayscale image, we have $L_k = 2^k - 1$). Signal to noise ratio quantifies the imperceptibility, by regarding the image as the signal and the message as the noise.

Here, we use a slightly different test-statistic, namely, Worst-case-Mean-Square-Error (WMSE) and the corresponding PSNR (per pixel) as our test-statistics. We define WMSE as follows:

If the secret data-bit is embedded in the i^{th} bitplane of a pixel, the worst-case error-square-per-pixel will be $= WSE = |W(i)(1 - 0)|^2 = (W(i))^2$, corresponding to when the corresponding bit in cover-image toggles in stego-image, after embedding the secret data-bit. For example, worst-case error-square-per-pixel for embedding a secret data-bit in the i^{th} bit plane in case of a pixel in classical

binary decomposition is $= (2^i)^2 = 4^i$, where $i \in Z^+ \cup \{0\}$. If the original k-bit grayscale cover-image has size $w \times h$, we define, $WMSE = w \times h \times (W(i))^2 = w \times h \times WSE$. Here, we try to minimize this WMSE (hence WSE) and maximize the corresponding PSNR. We use the results (3) and (12) to prove our following claims:

5.4.4 The proposed Prime Decomposition generates more (virtual) bit-planes

Using Classical binary decomposition, for a k-bit cover image, we get only k bit-planes per pixel, where we can embed our secret data bit. From (3) and (12), we get,

- $p_n = \theta(n \cdot \ln n)$
- $\exists \alpha_p \in \mathfrak{R}^+ : F_p(n) > (\alpha_p)^{n-1}, \alpha_p > \alpha_{p+1}, \forall p \in Z^+, \alpha_1 \approx 1.618$

Since $n \cdot \ln n = o(\alpha_p^n)$, it directly implies that $p_n = o(F_p(n))$. The maximum (highest) number that can be represented in n-bit number system using our prime decomposition is $\sum_{i=0}^{n-1} p_i$, and in case of n-bit number system using Fibonacci p-sequence decomposition is $\sum_{i=0}^{n-1} F_p(i)$. Now, it is easy to prove that, $\exists n_0 \in \mathfrak{N} : \forall n \geq n_0$ we have, $\sum_{i=0}^{n-1} F_p(i) > \sum_{i=0}^{n-1} p_i$.

Hence, using same number of bits it is possible to represent more numbers in case of the number system using Fibonacci-p-sequence decomposition, than that in case of the number system using prime decomposition, when number of bits is greater than some threshold. This in turn implies that number of virtual bit-planes generated in case of prime decomposition will be eventually (after some n) more than the corresponding number of virtual bit-planes generated by Fibonacci p-Sequence decomposition.

From the bar-chart shown in Figure-6, we see, for instance, to represent the pixel value 131, prime number system requires at least 12 bits, while for its Fibonacci counterpart 10 bits suffice. So, at the time of decomposition the same pixel value will generate 12 virtual bit-planes in case of prime decomposition and 10 for the later one, thereby increasing the space for embedding.

5.4.5 Prime Decomposition gives less distortion in higher bit-planes

Here, we assume the secret message length (in bits) is same as image size, for evaluation of our test statistics. For message with different length, the same can similarly be derived in a straight-forward manner.

In case of our Prime Decomposition, WMSE for embedding secret message bit only in l^{th} (virtual) bitplane of each pixel (after expressing a pixel in our prime number system, using prime decomposition technique) $= p_l^2$, because change in l^{th} bit plane of a pixel simply implies changing of the pixel value by at most l^{th} prime number.

From the above discussion and using equation (3), also treating image-size as constant we can immediately conclude, (for $l > 0$)

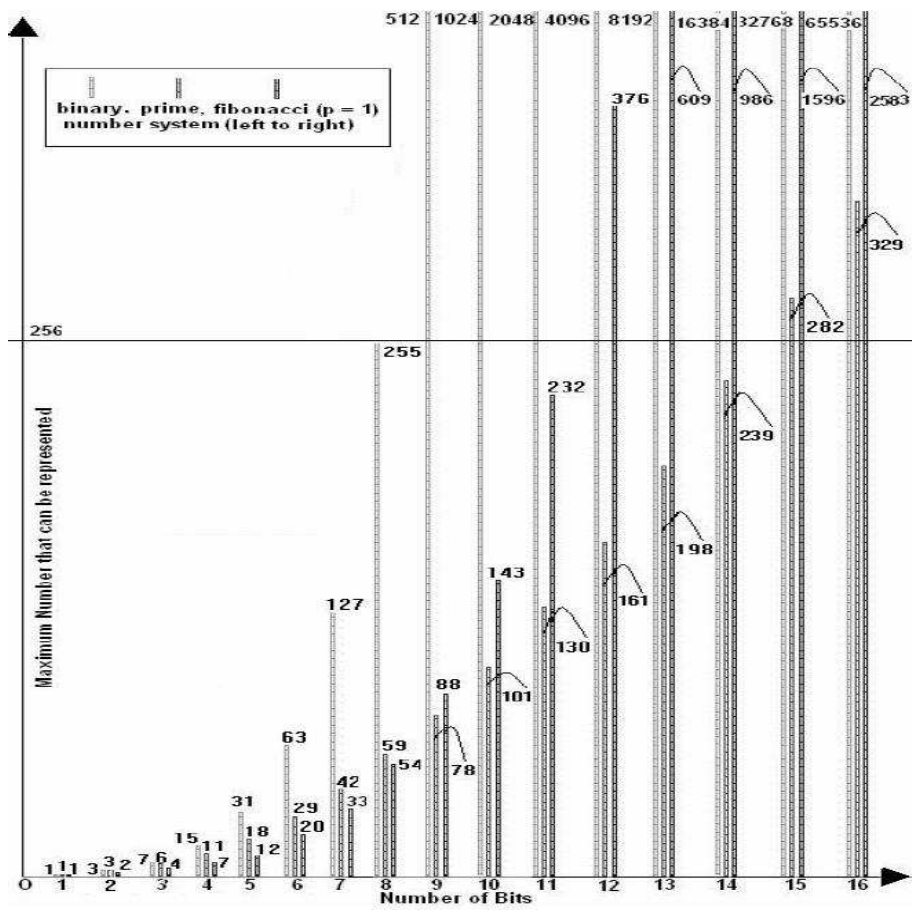


Figure 4: Maximum number that can be represented in different decomposition techniques

$$(WMSE_{l^{th} \text{ bitplane}})_{Prime-Decomposition} = w \times h \times p_l^2 = \theta(l^2 \cdot \text{Log}^2(l)). \quad (13)$$

whereas WMSE in case of classical (traditional) binary (LSB) data hiding technique is given by,

$$(WMSE_{l^{th} \text{ bitplane}})_{Classical-Binary-Decomposition} = \theta(4^l). \quad (14)$$

The above result implies that the distortion in case of prime decomposition is much less (since polynomial) than in case of classical binary decomposition (in which case it is exponential).

Now, let us calculate the WMSE for the embedding technique using Fibonacci p-sequence decomposition. In this case, WMSE for embedding secret message bit only in l^{th} (virtual) bit-plane of each pixel (after expressing it using Fibonacci-1-sequence decomposition) = $(F_p(l))^2$, because change in l^{th} plane of a pixel simply implies changing of the pixel value by at most l^{th} Fibonacci number.

From inequality (12), we immediately get that in case of $p = 1$, i.e., for the Fibonacci-1-sequence decomposition, we have,

$$(WMSE_{l^{th} \text{ bitplane}})_{Fibonacci-1-Sequence \text{ Decomposition}} = (F(l))^2 = \theta((2.618)^l)$$

Similarly, for other values of p , one can easily derive (by induction) some exponential lower-bounds, which are definitely better than the exponential bound obtained in case of classical binary decomposition, but still they are exponential in nature, even if the base of the exponential lower bound will decrease gradually with increasing p . So, we can generalize the above result by the following,

$$(WMSE_{l^{th} \text{ bitplane}})_{Fibonacci-p-Sequence \text{ Decomposition}} > \theta\left((\alpha_p^2)^l\right),$$

$$\alpha_p \in \mathbb{R}^+, \alpha_1 = \frac{1 + \sqrt{5}}{2},$$

$$\alpha_p^2 > \alpha_{p+1}^2, \forall p \in \mathbb{Z}^+.$$

The sequence α_p^2 is decreasing in p . Obviously, Fibonacci-p-sequence decomposition, despite being better than classical binary decomposition, is still exponential and causes much-more distortion in the higher bit-planes, than our prime decomposition, in which case WMSE is polynomial (and not exponential!) in nature. The plot shown in Figure-5 proves our claim, it vindicates the polynomial nature of the weight function in case of prime decomposition against the exponential nature of classical binary and Fibonacci decomposition.

So from all above discussion, we conclude that Prime Decomposition gives less distortion than its competitors (namely classical binary and Fibonacci Decomposition) while embedding secret message in higher bit-planes.

At a glance, results obtained for test-statistic WMSE, for our k-bit cover image,

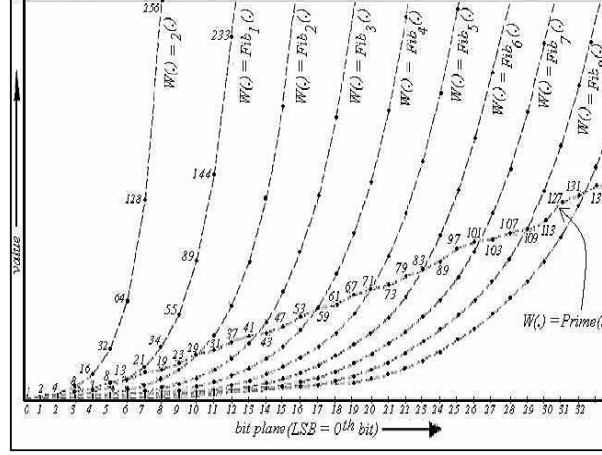


Figure 5: Weight functions for different decomposition techniques

$$\begin{aligned}
(WMSE_{l^{th} \text{ bitplane}})_{\text{Classical Binary Decomposition}} &= \theta(4^l). \\
(WMSE_{l^{th} \text{ bitplane}})_{\text{Prime Decomposition}} &= \theta(l^2 \cdot \log^2(l)). \\
(WMSE_{l^{th} \text{ bitplane}})_{\text{Fibonacci-}p \text{ Decomposition}} &= \theta((\alpha_p)^l), \\
\alpha_p \in \mathfrak{R}^+, 2.618 > \alpha_p > \alpha_{p+1}, \forall p \in Z^+, \text{ with} \\
\text{Fibonacci-1 Decomposition} &= \theta((2.618)^l)
\end{aligned} \tag{15}$$

Also, results for our test-statistic $PSNR_{worst}$,

$$\begin{aligned}
((PSNR_{worst})_{l^{th} \text{ bitplane}})_{\text{Binary Decomposition}} &= 10 \cdot \log_{10} \left(\frac{(2^k - 1)^2}{(2^l)^2} \right). \\
((PSNR_{worst})_{l^{th} \text{ bitplane}})_{\text{Prime Decomposition}} &= 10 \cdot \log_{10} \left(\frac{(2^k - 1)^2}{c \cdot l^2 \cdot \log^2(l)} \right), c \in \mathfrak{R}^+. \\
((PSNR_{worst})_{l^{th} \text{ bitplane}})_{\text{Fibonacci-}p \text{ Decomposition}} &= 10 \cdot \log_{10} \left(\frac{(2^k - 1)^2}{(\alpha_p)^l} \right), \\
\alpha_p \in \mathfrak{R}^+, 2.618 > \alpha_p > \alpha_{p+1}, \forall p \in Z^+, \text{ with} \\
((PSNR_{worst})_{l^{th} \text{ bitplane}})_{\text{Fibonacci-1 Decomposition}} &= 10 \cdot \log_{10} \left(\frac{(2^k - 1)^2}{(2.618)^l} \right).
\end{aligned} \tag{16}$$

6 Experimental Results for data-hiding technique using Prime decomposition

We have, as input:

- Cover Image: 8-bit (256 color) gray-level standard image of Lena.
- Secret message length = cover image size, (message string "sandipan" repeated multiple times to fill the cover image size).
- The secret message bits are embedded into one (selected) bit-plane per pixel only, the bitplane is indicated by the variable p .
- The test message is hidden into the chosen bitplane using different decomposition techniques, namely, the classical (traditional) binary (LSB) decomposition, Fibonacci 1-sequence decomposition and Prime decomposition separately and compared.

We get, as output:

- As was obvious from the above theoretical discussions, our experiment supported the fact that was proved mathematically.
- As obvious, as the relative entropy between the cover-image and the stego-image tends to be more and more positive (i.e., increases), we get more and more visible distortions in image rather than invisible watermark.
- Figure 6 illustrates gray level $[0 \dots 255]$ vs. frequency plot of the cover image and stego image in case of classical LSB data-hiding technique. As seen from the figure, we get only 8 bit-planes and the frequency distribution (as shown in histograms) and hence the probability mass function [27] corresponding to gray-level values changes abruptly, resulting in an increasing relative entropy between cover-image and stego-image, implying visible distortions, as we move towards higher bit-planes for embedding data bits.
- Figure 7 shows gray level $[0 \dots 255]$ vs. frequency plot of the cover image and stego image in case of data-hiding technique based on Fibonacci decomposition. This figure shows that, we get 12 bit-planes and the probability mass function corresponding to gray-level values changes less abruptly, resulting in a much less relative entropy between cover-image and stego-image, implying less visible distortions, as we move towards higher bit-planes for embedding data bits.
- Figure 8 again depicts gray level $[0 \dots 255]$ versus frequency plot of the cover image and stego image in case of data-hiding technique based on Prime decomposition. This figure shows that, we get 15 bit-planes and the change of frequency distribution (and hence probability mass function) corresponding to gray-level values is least when compared to the other two techniques, eventually resulting in a still less relative entropy between the cover image and stego-image, implying least visible distortions, as we move towards higher bitplanes for embedding data bits.

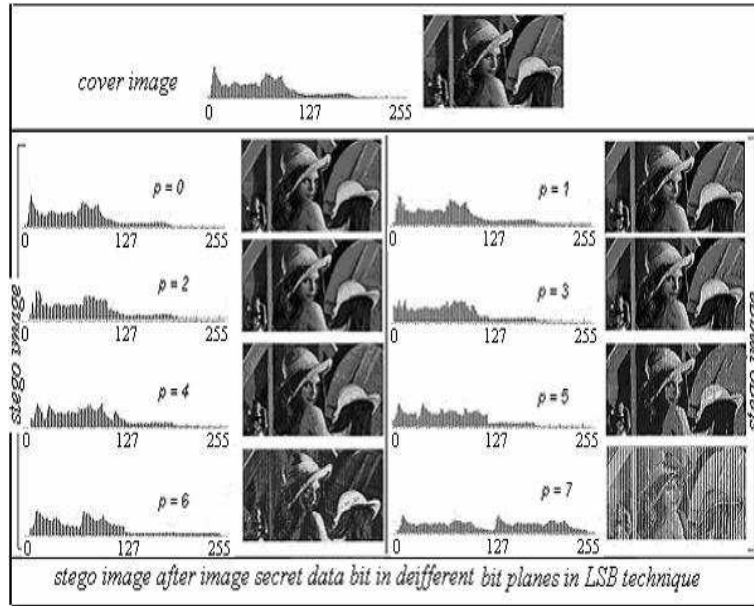


Figure 6: Frequency distribution of pixel gray-levels in different bit-planes before and after data-hiding in case of classical LSB technique

- Data-hiding technique using the prime decomposition has a better performance than that of Fibonacci decomposition, the later being more efficient than classical binary decomposition, when judged in terms of embedding secret data bit into higher bit-planes causing least distortion and thereby having least chance of being detected, since one of principal ends of data-hiding is to go as long as possible without being detected.
- Using classical binary decomposition, we get here only 8 bit planes (since an 8-bit image), using Fibonacci 1-sequence decomposition we have 12 (virtual) bit-planes, and using prime decomposition we have still higher, namely 15 (virtual) bit-planes.
- As evident in Figure 9, distortion is highest in case of classical binary decomposition, less prominent in case of Fibonacci, and least for prime.

This technique can be enhanced by embedding into more than one (virtual) bit-plane, following the variable-depth data-hiding technique [21].

7 The Natural Number Decomposition Technique

For further improvement in the same line, we introduce a new number system and use transformation into that in order to get more (virtual) bit-planes, and

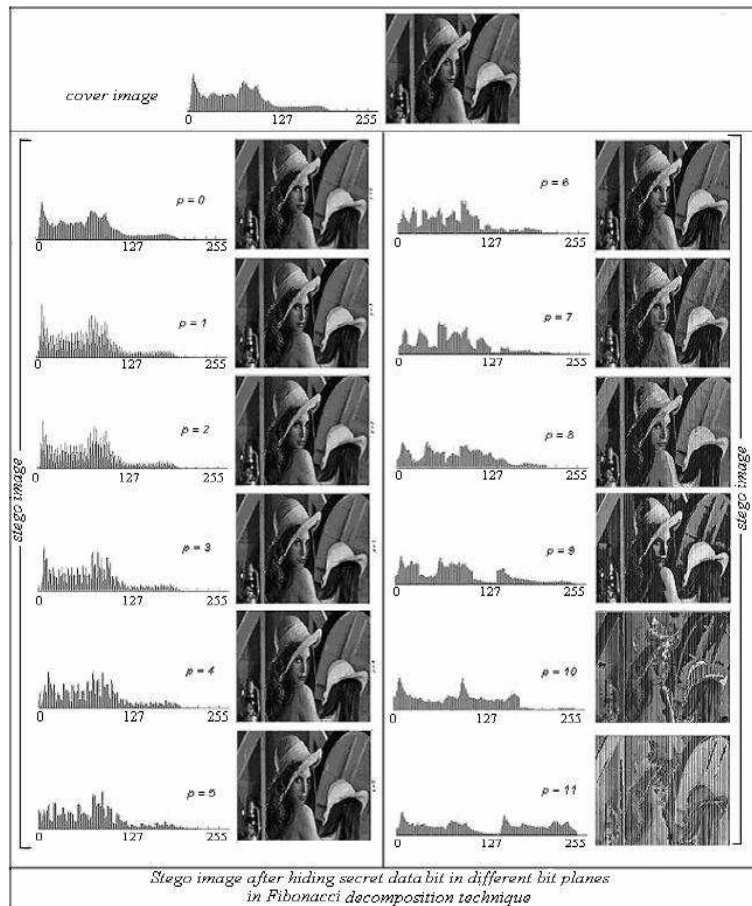


Figure 7: Frequency distribution of pixel gray-levels in different bit-planes before and after data-hiding in case of Fibonacci (1-sequence) decomposition technique

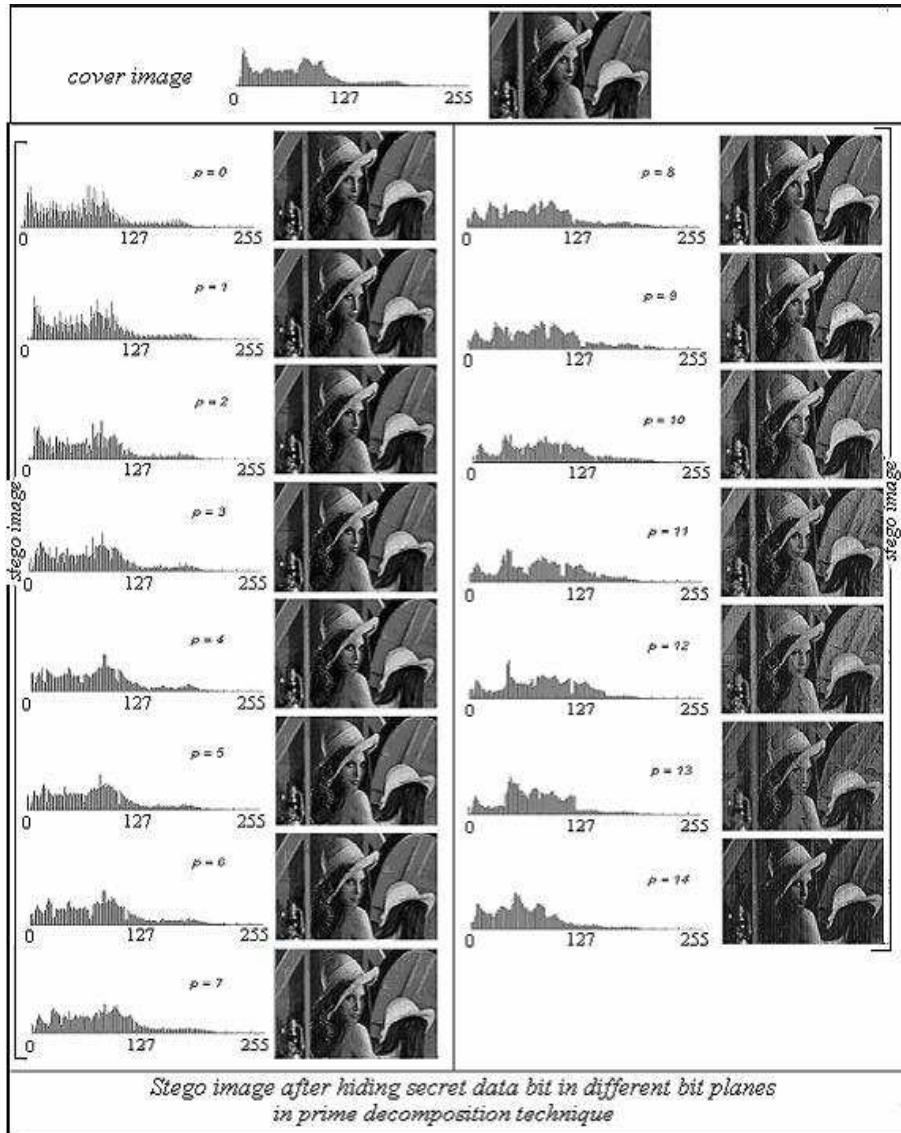


Figure 8: Frequency distribution of pixel gray-levels in different bit-planes before and after data-hiding in case of Prime decomposition technique

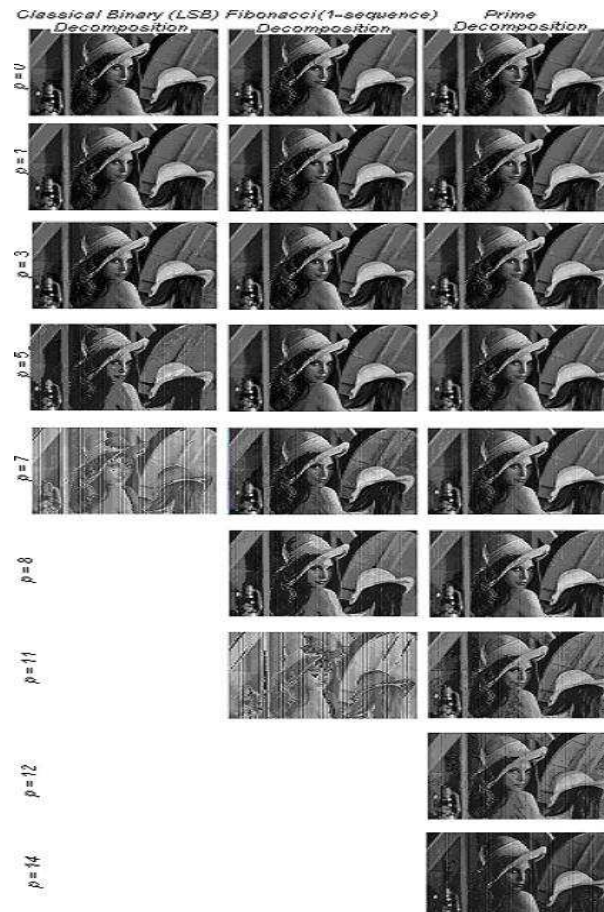


Figure 9: Result of embedding secret data in different bit-planes using different data-hiding techniques

also to have better image quality after embedding data into higher (virtual) bit-planes.

7.1 The Proposed Decomposition in Natural Numbers

We define yet another new number system, and as before we denote it as $(2, N(\cdot))$, where the weight function $N(\cdot)$ is defined as, $W(i) = N(i) = i+1, \forall i \in Z^+ \cup \{0\}$

Since the weight function here is composed of natural numbers, we name this number system as natural number system and the decomposition as natural number decomposition.

This technique also involves a lot of redundancy. Proving this is again very easy by using pigeonhole principle. Using n bits, we can have 2^n different binary combinations. But, as is obvious and we shall prove shortly that using n bits, all (and only) the numbers in the range $[0, n(n+1)/2]$, i.e., total $\frac{n(n+1)}{2} + 1$ different numbers can be represented using our natural number decomposition. Since by induction one can easily show, $2^n > \frac{n(n+1)}{2} + 1, \forall n \geq 2, n \in \mathbb{N}$, we conclude, by Pigeon hole principle that, at least 2 representations out of 2^n binary representations will represent the same value. Hence, we have redundancy.

As we need to make our transform one-to-one, what we do is exactly the same that we did in case of prime decomposition: if a number has more than one representation in our number system, we always take the lexicographically highest of them. (e.g., the number 3 has 2 different representations in 3-bit natural number system, namely, 100 and 011, since we have, $1.3 + 0.2 + 0.1 = 3$ and $0.3 + 1.2 + 1.1 = 3$. But, since 100 is lexicographically (from left to right) higher than 011, we choose 100 to be valid representation for 3 in our natural number system and thus discard 011, which is no longer a valid representation in our number system. $3 \equiv \max_{lexicographic}(100, 011) \equiv 100$ So, in our 3-bit example, the valid representations are: $000 \leftrightarrow 0, 001 \leftrightarrow 1, 010 \leftrightarrow 2, 100 \leftrightarrow 3, 101 \leftrightarrow 4, 110 \leftrightarrow 5, 111 \leftrightarrow 6$ Also, to avoid loss of message, we embed secret data bit to only those pixels, where, after embedding we get a valid representation in the number system. It is worth noticing that, up-to 3-bits, the prime number system and the natural number system are identical, after that they are different.

7.2 Embedding algorithm

- First, we need to find a number $n \in \mathbb{N}$ such that all possible pixel values in the range $[0, 2^k - 1]$ can be represented using first n natural numbers in our n -bit prime number system, so that we get n virtual bit-planes after decomposition. To find the n is quite easy, since we see, and we shall prove shortly that, in n -bit Natural Number System, all the numbers in the range $[0, n(n+1)/2]$ can be represented. So, our job reduces to finding an n such that $\frac{n(n+1)}{2} \geq 2^k - 1$, i.e., solving the following quadratic in-equality

$$n^2 + n - 2^{k+1} + 2 \geq 0,$$

$$\Rightarrow n \geq \frac{-1 + \sqrt{2^{k+3} + 9}}{2}, n \in \mathbb{Z}^+ \quad (17)$$

- After finding n , we create a map of k -bit (classical binary decomposition) to n -bit numbers (natural number decomposition), $n > k$, marking all the valid representations (as discussed in previous section) in our natural number system. For an 8-bit image the set of all possible pixel-values in the range $[0, 255]$ has the corresponding natural number decomposition as shown in Table-5.

For $k = 8$, we get,

$$n \geq \frac{-1 + \sqrt{2^{8+3} + 9}}{2} = \frac{-1 + \sqrt{2057}}{2} = \frac{44.35}{2} = 22.675 \Rightarrow n = 23$$

Hence, for an 8-bit image, we get 23 (virtual) bit-planes.

If we recapitulate our earlier result, as we see from the map shown in Table-2, in case of prime decomposition, it yields much less numbers of (virtual) bit planes (namely 15). Again it is noteworthy that the space to store the map is still increased. Although this computation of the map (one-time computation for a fixed value of k) is slightly more expensive and takes more space to store in case of our natural number decomposition than in case of prime decomposition, the first outperforms the later one when compared in terms of steganographic efficiency, i.e., in terms of embedded image quality, security (since number of virtual bit-planes will be more in case of the first) etc, as will be explained shortly.

- Next, for each pixel of the cover image, we choose a (virtual) bit plane, say p^{th} bit-plane and embed the secret data bit into that particular bit plane, by replacing the corresponding bit by the data bit, if and only if we find that after embedding the data bit, the resulting sequence is a valid representation in n -bit prime number system, i.e., exists in the map otherwise discard that particular pixel for data hiding.
- After embedding the secret message bit, we convert the resultant sequence in prime number system back to its value (in classical 8-4-2-1 binary number system) and we get our stego-image. This reverse conversion is easy, since we need to calculate $\sum_{i=0}^{n-1} b_i \cdot (i + 1)$ only, $b_i \in \{0, 1\}, \forall i \in \{0, n - 1\}$.

7.3 Extracting algorithm

The extraction algorithm is exactly the reverse. From the stego-image, we convert each pixel with embedded data bit to its corresponding natural decomposition and from the p^{th} bit-plane extract the secret message bit. Combine all the bits to get the secret message. Since, for efficient implementation, we shall have a hash-map for this conversion, the bit extraction is constant-time, so the secret message extraction will be polynomial (linear) in the length of the message embedded.

N	Natural Decomp	N	Natural Decomp
0	00000000000000000000	64	11001000000000000000
1	00000000000000000001	65	11010000000000000000
2	00000000000000000010	66	11100000000000000000
3	00000000000000000100	67	11100000000000000001
4	00000000000000001000	68	11100000000000000010
5	00000000000000010000	69	11100000000000000100
6	0000000000000100000	70	11100000000000001000
7	0000000000001000000	71	11100000000000010000
8	0000000000010000000	72	11100000000000100000
9	0000000000100000000	73	11100000000001000000
10	0000000001000000000	74	11100000000010000000
11	0000000010000000000	75	11100000000100000000
12	0000000100000000000	76	11100000001000000000
13	0000001000000000000	77	11100000010000000000
14	0000010000000000000	78	11100000100000000000
15	0000100000000000000	79	11100001000000000000
16	0001000000000000000	80	11100010000000000000
17	0010000000000000000	81	11100010000000000000
18	0001000000000000000	82	11100010000000000000
19	0001000000000000000	83	11100010000000000000
20	0010000000000000000	84	11100100000000000000
21	0010000000000000000	85	11101000000000000000
22	0100000000000000000	86	11110000000000000000
23	1000000000000000000	87	11110000000000000001
24	1000000000000000001	88	11110000000000000010
25	1000000000000000010	89	11110000000000000100
26	1000000000000000100	90	11110000000000001000
27	1000000000000001000	91	11110000000000010000
28	1000000000000010000	92	11110000000000100000
29	1000000000000100000	93	11110000000001000000
30	1000000000001000000	94	11110000000010000000
31	1000000000001000000	95	11110000000010000000
32	1000000000001000000	96	11110000000010000000
33	10000000000100000000	97	11110000000100000000
34	10000000001000000000	98	11110000001000000000
35	10000000010000000000	99	11110000010000000000
36	10000000100000000000	100	11110000100000000000
37	10000001000000000000	101	11110001000000000000
38	10000010000000000000	102	11110001000000000000
39	10000010000000000000	103	11110010000000000000
40	10000100000000000000	104	11110100000000000000
41	10001000000000000000	105	11111000000000000000
42	10001000000000000000	106	11111000000000000001
43	10010000000000000000	107	11111000000000000010
44	10100000000000000000	108	11111000000000000100
45	11000000000000000000	109	11111000000000000100
46	11000000000000000001	110	111110000000000010000
47	11000000000000000010	111	111110000000000100000
48	11000000000000000100	112	111110000000001000000
49	11000000000000001000	113	111110000000010000000
50	11000000000000010000	114	111110000000100000000
51	11000000000000100000	115	111110000000100000000
52	11000000000001000000	116	111110000001000000000
53	11000000000010000000	117	111110000010000000000
54	11000000000010000000	118	111110000100000000000
55	11000000000100000000	119	111110001000000000000
56	11000000001000000000	120	111110001000000000000
57	11000000010000000000	121	111110010000000000000
58	11000000100000000000	122	111110100000000000000
59	11000000100000000000	123	111111000000000000000
60	11000001000000000000	124	111111000000000000001
61	11000010000000000000	125	111111000000000000010
62	11000100000000000000	126	111111000000000000100
63	11001000000000000000	127	111111000000000001000

Table 5: Natural Number decomposition yielding 23 virtual bit-planes

7.4 The performance analysis : Comparison between Prime Decomposition and Natural Number Decomposition

In this section, we do a comparative study between the different decompositions and its effect upon higher-bit-plane data-hiding. We basically try to prove our following claims,

7.4.1 In k -bit Natural Number System, all the numbers in the range $[0, k(k+1)/2]$ can be represented and only these numbers can be represented

Proof by Induction on k :

Basis: $k = 1$, we can represent only 2 numbers, namely 0 and 1, but we have, $\frac{k(k+1)}{2} = 1$, i.e., all the numbers (and only these numbers) in the range $[0, 1]$, i.e., $[0, \frac{k(k+1)}{2}]$ can be represented for $k = 1$.

Induction hypothesis: Let us assume the above result holds $\forall k \leq n, n \in \mathbb{N}$.

Now, let us prove the same for $k = n + 1$.

From induction hypothesis, we know, using n bit Natural Number System, all (and only) the numbers in the range $[0, \frac{n(n+1)}{2}]$ can be represented. Let us list all the valid representations in n bit,

$$0 \equiv b_{0,n-1}b_{0,n-2} \dots b_{0,1}b_{0,0} \equiv 0000 \dots 00$$

$$1 \equiv b_{1,n-1}b_{1,n-2} \dots b_{1,1}b_{1,0} \equiv 0000 \dots 01$$

...

...

$$n(n+1)/2 \equiv b_{n(n+1)/2,n-1}b_{n(n+1)/2,n-2} \dots b_{n(n+1)/2,1}b_{n(n+1)/2,0} \equiv 1111 \dots 11$$

Now, for $(n+1)$ bit Natural Number System, we have the weight corresponding to the n^{th} significant Bit (MSB), $W(n) = n + 1$.

So when the MSB is 0, we have all the numbers in the range $[0, \frac{n(n+1)}{2}]$

$$0b_{0,n-1}b_{0,n-2}$$

$$0b_{1,n-1}b_{1,n-2}$$

...

...

$$0b_{n(n+1)/2,n-1}$$

and when the MSB is 1, we get a new set of $\frac{n(n+1)}{2} + 1$ numbers

$$n + 1 + 0,$$

$$n + 1 + 1,$$

$$n + 1 + 2,$$

...

...

$$n + 1 + \frac{n(n+1)}{2},$$

i.e., all the (consecutive) numbers in the range $[n + 1, \frac{(n+1)(n+2)}{2}]$.

$$\begin{aligned} &1b_{0,n-1}b_{0,n-2} \\ &1b_{1,n-1}b_{1,n-2} \\ &\dots \\ &\dots \\ &1b_{n(n+1)/2,n-1} \end{aligned}$$

So, we get all the numbers in the range $[0, \frac{n(n+1)}{2}] \cup [n + 1, \frac{(n+1)(n+2)}{2}] = [0, \frac{(n+1)(n+2)}{2}]$

Also, the maximum number that can be represented (all 1's) using $(n + 1)$ bit Natural Number System. $= (n + 1) + (n) + (n - 1) + \dots + (3) + (2) + (1) = \frac{(n+1)(n+2)}{2}$, and minimum number that can be represented (all 0's) is 0. Hence, only the numbers in this range can be represented.

Hence, we proved for $k = n + 1$ also. $\Rightarrow \forall k \in \mathbb{N}$ the above result holds.

7.4.2 The proposed Natural Number Decomposition generates more (virtual) bit-planes

Using Classical binary decomposition, for a k-bit cover image, we get only k bit-planes per pixel, where we can embed our secret data bit. From equation (3), we get, $p_n = \theta(n \cdot \ln(n))$ Since $n + 1 = o(n \cdot \ln(n))$, the weight corresponding to the n^{th} bit in our number system using natural number decomposition eventually becomes much higher than the weight corresponding to the n^{th} bit in the number system using prime decomposition. In n-bit Prime Number System, the numbers in the range $[0, \sum_{i=0}^{n-1} p_i]$ can be represented, while in our n-bit Natural Number System, the numbers in the range $[0, \sum_{i=0}^{n-1} (i + 1)] = [0, \sum_{i=1}^n i] = [0, \frac{n(n+1)}{2}]$ can be represented. Now, it is easy to prove that $\exists n_0 \in \mathbb{N} : \forall n \geq n_0$, we have, $\sum_{i=0}^{i=n-1} p_i > \frac{n(n+1)}{2}$.

Hence, using same number of bits, it is eventually possible to represent more numbers in case of the number system using prime decomposition, than that in case of the number system using natural number decomposition. This in turn implies that number of virtual bit-planes generated in case of natural number decomposition will be eventually more than the corresponding number of virtual bit-planes generated by prime decomposition.

From The bar-chart shown in Figure-10, we see that, in order to represent the pixel value 92, Natural number system requires at least 14 bits, while for Prime number system 10 bits suffice. So, at the time of decomposition the same pixel value will generate 14 virtual bit-planes in case of natural number decomposition and 10 for the prime, thereby increasing the space for embedding.

7.4.3 Natural Number Decomposition gives less distortion in higher bit-planes

Here we assume the secret message length (in bits) is same as image size, for evaluation of our test statistics. For message with different length, the same

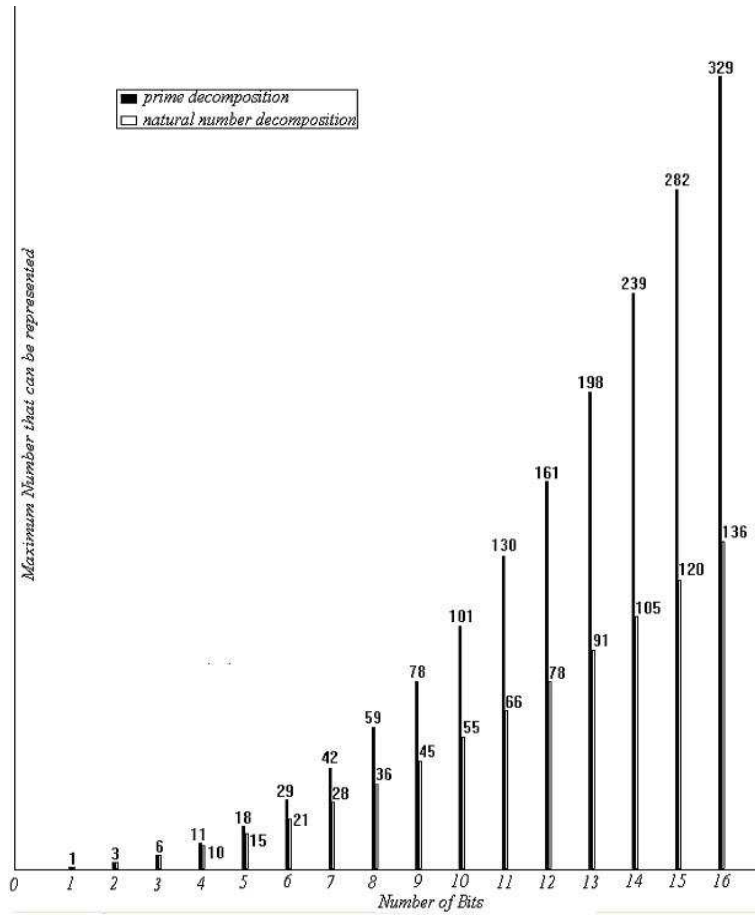


Figure 10: Maximum number that can be represented in prime and natural number decomposition techniques

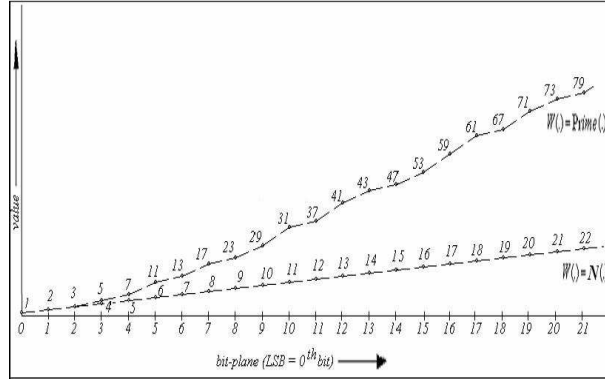


Figure 11: Weight functions for different decomposition techniques

can similarly be derived in a straight-forward manner.

In case of Prime Decomposition technique, WMSE for embedding secret message bit only in l^{th} (virtual) bitplane of each pixel (after expressing a pixel in our prime number system, using prime decomposition technique) = p_l^2 , because change in l^{th} bit plane of a pixel simply implies changing of the pixel value by at most the l^{th} prime number. From above, (treating image-size as constant) we can immediately conclude, from equation (3), for $l > 0$

$$(WMSE_{l^{th} \text{ bitplane}})_{Prime \text{ Decomposition}} = w \times h \times p_l^2 = \theta(l^2 \cdot \log^2(l))$$

In case of our Natural Decomposition, WMSE for embedding secret message bit only in l^{th} (virtual) bit-plane of each pixel (after expressing a pixel in our natural number system, using natural number decomposition technique) = $(l + 1)^2$. From above, (treating image-size as constant again) we can immediately conclude,

$$(WMSE_{l^{th} \text{ bitplane}})_{Natural \text{ Number Decomposition}} = (l + 1)^2 = \theta(l^2).$$

Since $(l + 1)^2 = o(l^2 \cdot \log^2(l))$, eventually we have,

$$(WMSE_{l^{th} \text{ bitplane}})_{Natural \text{ Decomposition}} < (WMSE_{l^{th} \text{ bitplane}})_{Prime \text{ Decomposition}}$$

The above result implies that the distortion in case of natural number decomposition is much less than that in case of prime decomposition. The plot shown in Figure-11 buttresses our claim, it compares the nature of the weight function in case of prime decomposition against that of the natural number decomposition.

So, from all above discussion, we conclude that Natural Number Decomposition gives less distortion than Prime Decomposition technique, while embedding secret message in higher bit-planes.

At a glance, results obtained for test-statistic WMSE, in case of our k-bit cover image,

$$\begin{aligned}
(WMSE_{l^{th} \text{ bitplane}})_{\text{Classical Binary Decomposition}} &= \theta(4^l). \\
(WMSE_{l^{th} \text{ bitplane}})_{\text{Prime Decomposition}} &= \theta(l^2 \cdot \log^2(l)). \\
(WMSE_{l^{th} \text{ bitplane}})_{\text{Natural Number Decomposition}} &= (l+1)^2 = \theta(l^2). \quad (18)
\end{aligned}$$

Also, results for our test-statistic $PSNR_{worst}$,

$$\begin{aligned}
((PSNR_{worst})_{l^{th} \text{ bitplane}})_{\text{Classical Binary Decomposition}} &= 10 \cdot \log_{10} \left(\frac{(2^k - 1)^2}{(2^l)^2} \right). \\
((PSNR_{worst})_{l^{th} \text{ bitplane}})_{\text{Prime Decomposition}} &= 10 \cdot \log_{10} \left(\frac{(2^k - 1)^2}{c \cdot l^2 \cdot \log^2(l)} \right). \\
((PSNR_{worst})_{l^{th} \text{ bitplane}})_{\text{Natural Number Decomposition}} &= 10 \cdot \log_{10} \left(\frac{(2^k - 1)^2}{(l+1)^2} \right). \quad (19)
\end{aligned}$$

From equations (18) and (19), we see that, WMSE gradually decreased from Binary to Prime and then from Prime to Natural decomposition techniques (minimized in case of Natural number decomposition), ensuring lesser probability of distortion, while PSNR gradually increased along the same direction (maximized in case of Natural number decomposition), implying more impercibility in message hiding.

7.4.4 Natural Number Decomposition is Optimal

This particular decomposition technique is optimal in the sense that it generates maximum number of (virtual) bit-planes and also least distortion while embedding in higher bit-planes, when the weight function is strictly monotonically increasing. Since, among all monotonic strictly increasing sequences of positive integers, natural number sequence is the tightest, all others are subsequences of the natural number sequence. Our generalized model indicates that the optimality of our technique depends on which number system we choose, or more precisely, which weight function we define. Since weight function $W : Z^+ \cup \{0\} \rightarrow Z^+$ (Since we are going to represent pixel-values, that are nothing but non-negative integers, the co-domain of our weight function is set of non-negative integers. Also, weight function is assumed to be one-one, otherwise there will be too much redundancy) is optimized when it is defined as $W(i) = i + 1, \forall i \in Z^+ \cup \{0\}$, i.e., in case of natural number decomposition.

Since we have, the weight function $W : Z^+ \cup \{0\} \rightarrow Z^+$, that assigns a bit-plane (index) an integral weight, if we assume that weight corresponding to a bit-plane is unique and the weight is monotonically increasing, one of the simplest but yet optimal way to construct such an weight function is to assign consecutive natural number values to the weights corresponding to each bit-plane, i.e., $W(i) = i + 1, \forall i \in Z^+ \cup \{0\}$ (We defined $W(i) = i + 1$ instead of $W(i) = i$, since we want all-zero representation for the value 0, in this particular number system). Now, this particular decomposition in virtual bit-planes and

embedding technique gives us optimal result. We get optimal performance of any data-hiding technique by minimizing our test-statistic WMSE. For embedding data in l^{th} virtual bit-plane, we have $(WMSE)_{l^{th} \text{ bitplane}} = (W(l))^2$, so minimizing WMSE implies minimizing the weight function $W(\cdot)$, but having our weight function allowed to assume integral values only, and also assuming the values assigned by W are unique (W is injective, we discard the un-interesting case when weight-values corresponding to more than one bit-planes are equal), we can without loss of generality assume W to be monotonically increasing. But, according to the above condition imposed on W , we see that such strictly increasing W assigning minimum integral weight-values to different bit planes must be linear in bit-plane index.

Put it in another way, for n -bit number system, we need n different weights that are to be assigned to weight-values corresponding to n bit-planes. But, the assigning must also guarantee that these weight values are minimum possible. Such n different positive integral values must be smallest n consecutive natural numbers, i.e., $1, 2, 3, \dots, n$. But, our weight function $W(i) = i + 1, \forall i \in Z^+ \cup \{0\}$ merely gives these values as weights only, hence this technique is optimal.

Using classical binary decomposition, we get k bit planes only corresponding to a k -bit image pixel value, but in case of natural number decomposition, we get, n -bit pixels, where n satisfies,

$$\begin{aligned} n^2 + n - 2^{k+1} + 2 &\geq 0 \\ \Rightarrow n &\geq \frac{-1 + \sqrt{2^{k+3} + 9}}{2}, n \in Z^+ \\ &\Rightarrow n = \theta(2^{\frac{k}{2}}) \end{aligned} \tag{20}$$

8 Experimental Results for Natural Number decomposition technique

We have, again, as input:

- Cover Image: 8-bit (256 color) gray-level standard image of Lena.
- Secret message length = cover image size, (message string "sandipan" repeated multiple times to fill the cover image size).
- The secret message bits are embedded into one (selected) bit-plane per pixel only, the bitplane is indicated by the variable p .
- The test message is hidden ([26]) into the chosen bit-plane using different decomposition techniques, namely, the classical (traditional) binary (LSB) decomposition, Fibonacci 1-sequence decomposition and Prime decomposition separately and compared.

We get, as output:

- As was obvious from the above theoretical discussions, our experiment supported the fact that was proved mathematically, i.e., we got more (virtual) bit-planes and less distortion after embedding secret message into the bit-planes in case of Natural and Prime decomposition technique than in case of Fibonacci technique and classical binary LSB data hiding technique. We could also capture the hidden message from the stego-image successfully using our decoding technique.
- As obvious, as the relative entropy between the cover-image and the stego-image tends to be more and more positive (i.e., increases), we get more and more visible distortions in image rather than invisible watermark.
- As recapitulation of our earlier experimental result, Figure-8 shows gray level (0...255) vs. frequency plot of the cover image and stego-image in case of data-hiding technique based on Prime decomposition. This figure shows that, we get 15 bit-planes and the change of frequency distribution (and hence probability mass function) corresponding to graylevel values is least when compared to the other two techniques, eventually resulting in a still less relative entropy between the cover-image and stego-image, implying least visible distortions, as we move towards higher bit-planes for embedding data bits.
- Figure-12 shows gray level (0...255) vs. frequency plot of the cover image and stego image in case of data-hiding technique based on Natural Number decomposition. We get 23 bit-planes and the change of frequency distribution (and hence probability mass function) corresponding to gray-level values is least when compared to the other two techniques, eventually resulting in a still less relative entropy between the cover-image and stego-image, implying least visible distortions, as we move towards higher bit-planes for embedding data bits.
- Data-hiding technique using the natural number decomposition has a better performance than that of prime decomposition, the later being more efficient than classical binary decomposition, when judged in terms of embedding secret data bit into higher bit-planes causing least distortion and thereby having least chance of being detected, since one of principle ends of data-hiding is to go as long as possible without being detected.
- Using classical binary decomposition, we get here only 8 bit planes (since an 8 bit image), using Fibonacci 1-sequence decomposition we have 12 (virtual) bit-planes, and using prime decomposition we have 15 (virtual) bit-planes, but using natural decomposition, we have the highest, namely, 23 (virtual) bit planes.
- As vindicated in the figures 8 and 9, distortion is much less for natural decomposition, than that in case of prime. This technique can also be

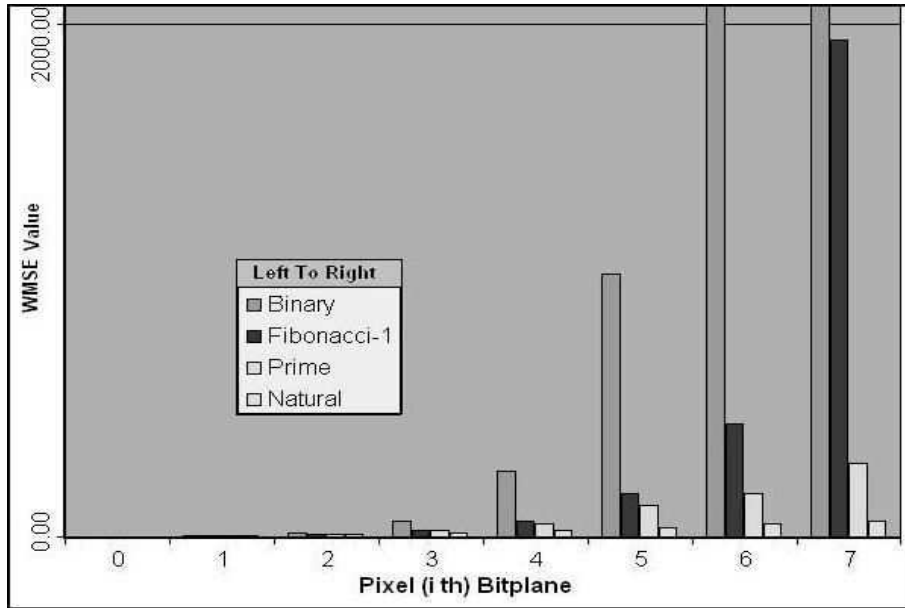


Figure 13: Comparison of WMSE values for different data hiding techniques

enhanced by embedding into more than one (virtual) bit-plane, following the variable-depth data-hiding technique [9].

- Figures 13 and 14 show comparison of WMSE and PSNR values, respectively, obtained from experimental results. It clearly shows that even for higher bitplanes the secret data can be reliably hidden with quite high PSNR value. Hence, it will be difficult for the attacker to predict the secret embedding bitplane.
- The experimental results were obtained by implementing the algorithms and data hiding techniques in C++ (open source gcc) and (gray-scale) Lena bitmap as input image file. Also the extraction algorithms that described for both the techniques run at linear time in length of message embedded.

9 Conclusions

This chapter presented very simple methods of data hiding technique using prime numbers / natural numbers. It is shown (both theoretically and experimentally) that the data-hiding technique using prime decomposition outperforms the famous LSB data hiding technique using classical binary decomposition and that using Fibonacci p-sequence decomposition. Also, the technique



Figure 14: Comparison of PSNR values for different data hiding techniques

using natural number decomposition outperforms the one using prime decomposition, when thought with respect to embedding secret data bits at higher bit-planes (since number of virtual bit-planes generated also increases) with less detectable distortion. We have shown all our experimental results using the famous Lena image, but since in all our theoretical derivation above we have shown our test-statistic value (WMSE, PSNR) independent of the probability mass function of the gray levels of the input image, the (worst-case) result will be similar if we use any gray-level image as input, instead of the Lena image.

References

- [1] F. Battisti, M. Carli, A. Neri, K. Egiazarian, "A Generalized Fibonacci LSB Data Hiding Technique", 3rd International Conference on Computers and Devices for Communication (CODEC- 06) TEA, Institute of Radio Physics and Electronics, University of Calcutta, December 18-20, 2006.
- [2] N. Nikolaidis and I. Pitas, "Robust image watermarking in the spatial domain", Signal Processing, vol. 66, no. 3, pp. 385-403, May 1998.
- [3] R. Wolfgang and E. Delp, "A watermark for digital images", in IEEE Proc. Int. Conf. Image Proc. ICIP 1996, pp. 219-222.

- [4] R. Z. Wang, C. F. Lin and I. C. Lin, "Image Hiding by LSB substitution and genetic algorithm", *Pattern Recognition*, Vol. 34, No. 3, pp. 671-683, 2001.
- [5] D. Gruhl, W. Bender, and N. Morimoto, "Techniques for data hiding", Tech. Rep., MIT Media Lab, Cambridge, MA, 1994.
- [6] M. Barni, F. Bartolini, V. Cappellini, and A. Piva, "A dct-domain system for robust image watermarking", *Signal Processing*, vol. 66, no. 3, pp. 357-372, May 1998.
- [7] I. Cox, J. Kilian, F. Leighton, and T. Shamoan, "Secure spread spectrum watermarking for multimedia", *IEEE Transaction on Image Processing*, vol. 6, no. 12, December 1997.
- [8] H. Inoue, A. Miyazaki, and T. Katsura, "An image watermarking method based on the wavelet transform", in *Proceedings of IEEE International Conference on Image Processing*, Kobe, Japan, October 1999, pp. 296-300.
- [9] C. Shao-Hui, Y. Tian-Hang, G. Hong-Xun, Wen, "A variable depth LSB data hiding technique in images", in *Proc. Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*, Vol. 7, 26-29 August 2004, pp. 3990 - 3994.
- [10] A. Horadam, "A generalized Fibonacci sequence", *American Mathematical Monthly*, no. 68, pp 455-459,1961.
- [11] Jr. Hoggatt Verner E., "Fibonacci and Lucas numbers", *The Fibonacci Association*, Santa Clara, California, USA, 1972.
- [12] D. De Luca Picione, F. Battisti, M. Carli, J. Astola, and K. Egiazarian, "A Fibonacci LSB data hiding technique", *Proc. European signal processing conference*, 2006.
- [13] Basin, S. L. and Hoggatt, V. E. Jr., "A Primer on the Fibonacci Sequence", *Fib. Quart.* 1, 1963
- [14] M. Kutter and S. Winkler, "A vision-based masking model for spread-spectrum image watermarking", *IEEE Trans. on Image Processing*, vol. 11, no. 1, pp. 16-25, January 2002.
- [15] Fabien A.P.Petitcolas, Ross J. Anderson and Markus G. Kubn, "Information hiding - a survey", *Proceedings of the IEEE Special issue on protection of multimedia content*, Vol.87, No. 7, pp. 1062-1078, July 1999
- [16] A. Westfield, A. Pfitzmann, "Attacks on steganographic system", *Proceeding of Information Hiding*, Springer-Verlag. LNCS 1768, pp. 61-76, 1999
- [17] R. Z. Wang, C. F. Lin and I. C. Lin, "Hiding data in images by optimal moderately-significant-bit replacement", *IEE Electronics Letters*, Vol. 37, No.25, pp. 2069-2070, December 2000.

- [18] Chi-Kwong Chan, L.M. Cbeng, “Hiding data in images by simple LSB substitution”, *Pattern Recognition*, Vol. 37, pp. 469-474, 2004.
- [19] Jessica Fridrich, Miroslav Goljan and Rui Du., “Detecting LSB steganography in color and grayscale images”, *Magazine of IEEE Multimedia Special Issue on Security*, up. 22-28, October- December 2001.
- [20] Shah Triiehi and R. Chandramouli, “Active steganalysis of sequential steganography”, *Proceeding of SPIE-IS & T Electronic Imaging, Security and watermarking of multimedia contents V*, SPIE Vo1.5020, pp. 123-130, 2003.
- [21] Shao-Huiliv, Tian-Hang Chen, Hong-Xun Yao, Wen Gao, “A Variable Depth LSB Data Hiding Technique in images”, *Proceedings of the Third International Conference on Machine Learning and Cybernetics*, Shanghai, 26-29 August 2004.
- [22] A. K. Jain, “Advances in mathematical models for image processing”, *Proceedings of the IEEE*, 69(5), pp. 502-528, May 1981.
- [23] M. Ramkumar and A. N. Akansu, “Theoretical capacity measures for data hiding in compressed images”, *In Proc. SPIE, Voice, Video and Data Communications*, volume 3528, pp. 482-492, November 1998.
- [24] R. Machado, “Stego”, <http://www.nitv.net/mech/Romana/stego.html> (1994).
- [25] W. Bender, “Data Hiding”, *News in the Future*, MIT Media Laboratory, unpublished lecture notes (1994).
- [26] K. Matsui and K. Tanaka, “Video-Steganography: How to Secretly Embed a Signature in a Picture”, *IMA Intellectual Property Project Proceedings* (1994).
- [27] A. V. Drake, “*Fundamentals of Applied Probability*”, McGraw-Hill, Inc., New York (1967).
- [28] I. Pitas, T. H. Kaskalis, “Applying signatures on Digital Images”, *IEEE Workshop on Nonlinear Image and Signal Processing*, Neos Marmaras, Greece, pp. 460-463, June 1995
- [29] N. Nikolaidis, I. Pitas, “Copyright protection of images using robust digital signatures”, *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP-96)*, vol. 4, pp. 2168- 2171, May 1996.
- [30] E. Koch, J. Zhao, “Towards robust and hidden image copyright labeling”, *Proc. of IEEE Workshop on Nonlinear Signal and Image Processing*, Neos Marmaras, Greece, pp. 452-455, 20-22 June 1995.

- [31] J. Zhao, E. Koch, “Embedding robust labels into images for copyright protection”, Technical report, Fraunhofer Institute for Computer Graphics, Darmstadt, Germany, 1994.
- [32] C. Hsu, J. Wu, “Hidden-signatures in images”, Proc. 1996 IEEE Int. Conference on Image Processing (ICIP 96), vol. 3, pp. 219-222.
- [33] B. Pfitzmann, “Information hiding terminology”, in Information Hiding, First International Workshop (R. Anderson, ed.), vol. 1174 of Lecture Notes in Computer Science, Springer, 1996.
- [34] M. Naor, A. Shamir, “Visual Cryptography”, in Advances in Cryptology: EUROCRYPT’94 (A. De Santis ed.) vol. 950 of Lecture Notes in Computer Science, pp. 1-12, Springer, 1995.
- [35] “Information hiding”, Series of International Workshops, 1996-2002. Proceedings in Lecture Notes in Computer Science, Springer.
- [36] V. Bhaskaran, K. Konstantinides, “Image and Video Compression Standards: Algorithms and Architectures”, Kluwer, 1995.
- [37] Christian Cachin, “An Information-Theoretic Model for Steganography”, Lecture Notes in Computer Science, vol. 1525, Springer.
- [38] T. M. Cover and J. A. Thomas, Elements of Information Theory, Wiley-Interscience, 1991.
- [39] Telang S. G., Number Theory, Tata McGraw-Hill, ISBN 0-07-462480-6, First Reprint, 1999, pp. 617-631.
- [40] Dey S., Abraham A., Sanyal S., An LSB Data Hiding Technique Using Natural Numbers, IEEE Third International Conference on Intelligent information hiding and Multimedia Signal Processing, Taiwan, IEEE Computer Society press, USA, ISBN 0-7695-2994-1, pp. 473-476, 2007.
- [41] Dey S., Abraham A., Sanyal S., An LSB Data Hiding Technique Using Prime Numbers, Third International Symposium on Information Assurance and Security, IEEE Computer Society press, USA, ISBN 0-7695-2876-7, pp. 101-106, 2007.

10 Short CV of the Authors

Sandipan Dey Sandipan Dey is a Technical Lead in Cognizant Technology Solutions, India. He has four years of experience in the software industry. He has C/C++ application development experience as well as research and development experience. He received his B.E. from Jadavpur University, India. His research interests include Computer Security, Steganography, Cryptography, Algorithm. He is also interested in Compilers, program analysis and evolutionary algorithms. He has published several papers in international journals and conferences.

Ajith Abraham Dr. Ajith Abraham's research and development experience includes over 17 years in the Industry and Academia spanning different continents in Australia, America, Asia and Europe. He works in a multidisciplinary environment involving computational intelligence, network security, sensor networks, e-commerce, Web intelligence, Web services, computational grids, data mining and applied to various real world problems. He has authored/co-authored over 350 refereed journal/conference papers and book chapters and some of the works have also won best paper awards at international conferences and also received several citations. Some of the articles are available in the ScienceDirect Top 25 hottest articles. His research interests in advanced computational intelligence include Nature Inspired Hybrid Intelligent Systems involving connectionist network learning, fuzzy inference systems, rough set, swarm intelligence, evolutionary computation, bacterial foraging, distributed artificial intelligence, multi-agent systems and other heuristics. He has given more than 20 plenary lectures and conference tutorials in these areas. Currently, he is working with the Norwegian University of Science and Technology (NTNU), Norway. Before joining NTNU, he was working under the Institute for Information Technology Advancement (IITA) Professorship Program funded by the South Korean Government. He was a Researcher at Rovira i Virgili University, Spain during 2005-2006. He also holds an Adjunct Professor appointment in Jinan University, China and Dalian Maritime University, China. He has held academic appointments in Monash University, Australia; Oklahoma State University, USA; Chung-Ang University, Seoul and Yonsei University, Seoul. Before turning into a full time academic, he was working with three International companies: Keppel Engineering, Singapore, Hyundai Engineering, South Korea and Ashok Leyland Ltd., India where he was involved in different industrial research and development projects for nearly 8 years. He received Ph.D. degree in Computer Science from Monash University, Australia and a Master of Science degree from Nanyang Technological University, Singapore. He serves the editorial board of over 30 reputed International journals and has also guest edited 28 special issues on various topics. He is actively involved in the Hybrid Intelligent Systems (HIS) ; Intelligent Systems Design and Applications (ISDA) and Information Assurance and Security (IAS) series of International conferences. He is a Senior Member of IEEE (USA), IEEE Computer Society (USA), IET (UK), IEAust (Australia) etc. In 2008, he is the *General Chair/Co-chair* of Tenth International Conference on Computer Modeling and Simulation, (UKSIM'08), Cambridge, UK; Second Asia International Conference on Modeling and Simulation (AMS'08), Kuala Lumpur, Malaysia; Eight International Conference on Intelligent Systems Design and Applications (ISDA'08), Kaohsiung, Taiwan; Fourth International Symposium on Information Assurance and Security (IAS'08), Naples, Italy; 2nd European Symposium on Computer Modeling and Simulation, (EMS'08), Liverpool, UK; Eighth International Conference on Hybrid Intelligent Systems (HIS'08), Barcelona, Spain; Fifth International Conference on Soft Computing as Transdisciplinary Science and Technology (CSTST'08), Paris, France *Program Chair/Co-chair* of Third International Conference on Digital Information Management (ICDIM'08), Lon-

don, UK; 7th Computer Information Systems and Industrial Management Applications (CISIM'08), Ostrava, Czech Republic; Second European Conference on Data Mining (ECDM'08), Amsterdam, Netherlands and the *Tutorial Chair* of 2008 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT'08), Sydney, Australia More information at: <http://www.softcomputing.net>

Bijoy Bandopadhyay Dr. Bijoy Bandyopadhyay is currently a faculty member of the Department of Radio Physics and Electronics, University of Calcutta. He received his PhD, M. Tech. B.Tech. and B.Sc. degrees from University of Calcutta, Kolkata. His current research interest includes Microwave Tomography, Ionosphere Tomography, Atmospheric Electricity Parameters, and Computer Security (Steganography). He has published numerous papers in International Journals and Conferences.

Sugata Sanyal Dr. Sugata Sanyal is a Professor in the School of Technology and Computer Science at the Tata Institute of Fundamental Research, India. He received his Ph.D. degree from Mumbai University, India, M. Tech. from IIT, Kharagpur, India and B.E. from Jadavpur University, India. His current research interests include Multi-Factor Security Issues, Security in Wireless and Mobile Ad Hoc Networks, Distributed Processing, and Scheduling techniques. He has published numerous papers in national and international journals and attended many conferences. He is in the editorial board of four International Journals. He is co-recipient of Vividhlaxi Audyogik Samsodhan Vikas Kendra Award (VASVIK) for Electrical and Electronics Science and Technologies (combined) for the year 1985. He was a Visiting Professor in the Department of Electrical and Computer Engineering and Computer Science in the University of Cincinnati, Ohio, USA in 2003. He delivered a series of lectures and also interacted with the Research Scholars in the area of Network Security in USA, in University of Cincinnati, University of Iowa, Iowa State University and Oklahoma State University. He has been an Honorary Member of Technical Board in UTI (Unit Trust of India), SIDBI (Small Industries Development Bank of India) and Coal Mines Provident Funds Organization (CMPFO). He has also been acting as a consultant to a number of leading industrial houses in India. More information about his activities is available at <http://www.tifr.res.in/sanyal>. Sugata Sanyal is an honorary member of the Technical Board and has also served as a consultant: (Few significant ones): UTI (Unit Trust of India); SIDBI (Small Industries Development Bank of India); CMPFO (Coal Mines Provident Funds Organization); MAHAGENCO; Centre for Development of Telematics (CDOT); Crompton Greaves Limited; Tata Electronic Co. (Research and Development).