ШИФРОВАНИЕ ЦВЕТНОГО ИЗОБРАЖЕНИЯ НА ОСНОВЕ ФРАКТАЛОВ НЬЮТОНА И ЧЕБЫШЕВА С ИСПОЛЬЗОВАНИЕМ КОМПЛЕСНЫХ ГИПЕР-ДУАЛЬНЫХ ЧИСЕЛ

Олифер В.И.

Аннотация

Предложен алгоритм шифрования цветных изображений, основанный на использовании фракталов Ньютона и Чебышёва, построенных в пространстве комплексных гипер-дуальных чисел. Метод сочетает перестановку пикселей, нелинейное табличное преобразование и маскирование изображения фрактальным ключом. В отличие от известных подходов, генерация ключевых последовательностей осуществляется на основе итерационных методов Ньютона и Чебышёва, реализованных в алгебре комплексных гипер-дуальных чисел. Использование гипер-дуальной арифметики позволяет одновременно строить фрактальные структуры и выполнять автоматическое дифференцирование, обеспечивая получение первых и вторых производных с машинной точностью. Проведённые эксперименты включают анализ гистограмм и исследование чувствительности алгоритма на основе метрики NPCR. Результаты показывают высокую эффективность предложенного подхода для защиты цветных изображений.

*Ключевые слова***:** шифрование изображений, фрактал Ньютона, фрактал Чебышёва, комплексные гипер-дуальные числа, автоматическое дифференцирование, криптография изображений.

1. Введение

Защита цифровых изображений представляет собой важную задачу современной криптографии. В отличие от текстовых данных изображения обладают значительным объёмом, высокой избыточностью и сильной корреляцией между соседними пикселями. Поэтому разработка специализированных методов шифрования изображений остаётся актуальным направлением исследований.

Одним из перспективных подходов является использование хаотических и фрактальных систем. Особый интерес представляют фракталы, возникающие при применении итерационных методов поиска корней. Классическим примером являются фракталы Ньютона, образующие сложные бассейны притяжения различных корней функции [1].

В последние годы активно исследуются динамические свойства фракталов, возникающих при использовании методов более высокого порядка, включая метод Чебышёва третьего

порядка сходимости. Такие методы генерируют сложные множества Жюлиа и бассейны притяжения, обладающие развитой фрактальной структурой [2].

В настоящей работе предлагается использовать фракталы Ньютона и Чебышёва в пространстве комплексных гипер-дуальных чисел для построения криптографического алгоритма шифрования цветных изображений.

2. Комплексные гипер-дуальные числа и автоматическое дифференцирование

Комплексное гипер-дуальное число определяется выражением [3]

$$X = x + x_1\boldsymbol{\varepsilon} + x_2\boldsymbol{\omega}, \qquad (x, x_1, x_2) \in \mathbb{C},$$

где: $\boldsymbol{\varepsilon}$ и $\boldsymbol{\omega}$ – нильпотентные элементы, удовлетворяющие соотношениям, $\boldsymbol{\varepsilon^2} = 2\boldsymbol{\omega}$, $\boldsymbol{\varepsilon\omega} = \boldsymbol{\omega^2} = 0$.
Для двух элементов $X = x + x_1\boldsymbol{\varepsilon} + x_2\boldsymbol{\omega}$ и $Y = y + y_1\boldsymbol{\varepsilon} + y_2\boldsymbol{\omega}$ операции сложения и вычитания определяются покомпонентно:

$$X \pm Y = x \pm y + (x_1 \pm y_1)\boldsymbol{\varepsilon} + (x_2 \pm y_2)\boldsymbol{\omega} \tag{2.0}$$

Операция умножения задаётся правилом:

$$X{\cdot}Y = x{\cdot}y + (x{\cdot}y_1 + y{\cdot}x_1)\boldsymbol{\varepsilon} + (x{\cdot}y_2 + 2x_1{\cdot}y_1 + y{\cdot}x_2)\boldsymbol{\omega} \tag{2.1}$$

Пусть $F(X)$ – функция гипер-дуального комплексного аргумента.
Разложение $F(X)$ в ряд Тейлора с точностью до второго порядка даёт

$$F(X) = f(x) + x_1{\cdot}f'(x)\boldsymbol{\varepsilon} + (x_2{\cdot}f'(x) + x_1^2{\cdot}f''(x))\boldsymbol{\omega}, \tag{2.2}$$

где $f(x) = F(x + 0\boldsymbol{\varepsilon} + 0\boldsymbol{\omega})$, $f'(x) = \frac{df(x)}{dx}$, $f''(x) = \frac{d^2f(x)}{dx^2}$

При выборе $x_1 = 1$ и $x_2 = 0$ выражение (2.2) принимает вид:

$$F(X) = f(x) + f'(x)\boldsymbol{\varepsilon} + f''(x)\boldsymbol{\omega} \tag{2.3}$$

что позволяет получать точные значения первых и вторых производных без применения конечных разностей и символьного дифференцирования [4, 5].

Данное свойство является ключевым для реализации фракталов Ньютона и Чебышёва.

3. Фрактал Ньютона

Для произвольной аналитической функции $f(z)$ итерационный процесс Ньютона определяется формулой

$$z_{n+1} = z_n - f(z_n)\ /f'(z_n) \tag{3.1}$$

При построении фрактала каждая точка комплексного гипер-дуального пространства используется как начальное приближение. Цвет пикселя определяется номером корня и числом итераций, необходимых для достижения сходимости.

Бассейны притяжения различных корней образуют сложную самоподобную структуру, называемую фракталом Ньютона.

4. Фрактал Чебышёва

Для повышения сложности ключевого пространства дополнительно используется метод Чебышёва третьего порядка:

$$z_{n+1} = z_n - f(z_n)\ /f'(z_n)\ -\ \ f^2(z_n)f''(z_n)/2f'(z_n)^3 \tag{4.1}$$

В отличие от метода Ньютона здесь присутствует вторая производная функции, вычисляемая автоматически при помощи гипер-дуальной арифметики.

Фракталы Чебышёва характеризуются более сложной структурой границ бассейнов притяжения и высокой чувствительностью к начальным условиям.

5. Генерация фрактального ключа

Ключевая карта должна формироваться процедурой

fractalKey = **GenerateFractalKey**(w, h, centerX, centerY, *maxIter, F*)

где:

- w , h – размеры изображения;
- centerX, centerY – центр изображения;
- maxIter – максимальное число итераций;
- F – оператор Ньютона или Чебышёва.

Для каждой точки вычисляются:

- номер достигнутого корня;
- число выполненных итераций;
- значения гипер-дуальных компонент;

- После нормализации формируется ключевая матрица $K(x, y) \in [0, 255]$

6. Алгоритм шифрования

Предлагаемый алгоритм включает следующие последовательные этапа.

6.1 Шифрование (Encode)

Шаг	Описание
1. Input	original, seed, maxIter, F
2. Cipher1	cipher1 = **EncodeCipher1**(original, seed) – стохастическое побайтное преобразование
3. S-box	table **= GenerateCryptoTable16x16**(seed) – генерация таблицы подстановок
4. Cipher2	cipher2 **= EncodeCipher2**(cipher1, table) – нелинейная перестановка нибблов
5. Fractal Key	fractalKey = **GenerateFractalKey**(w,h,0,0,maxIter,F) – фрактальный ключ
6. Cipher3	cipher3 = **EncodeCipher3**(cipher2, fractalKey) – XOR с фрактальной картой

Результат: Cipher3 – финальный шифр исходного изображения, визуально представляющий собой белый шум.

6.2 Дешифрование (Decode)

Шаг	Описание
1. Input	cipher3, seed, maxIter, F
2. Fractal Key	fractalKey **= GenerateFractalKey**(_cipher3.Width, cipher3.Height, 0, 0, _maxIter, _F)
3. Cipher2	cipher2 = **DecodeCipher3**(cipher3, fractalKey)
4. S-box	table = **GenerateCryptoTable16x16**(seed)
5. Inverse S-box	invTable = **BuildInverseTable**(table)
6. Cipher1	cipher1 = **DecodeCipher2**(cipher2, invTable)
7. Original	original = **DecodeCipher1**(cipher1, _seed)

Алгоритм полностью обратим при корректном ключе.

7. Компьютерная реализация

Реализация методов Ньютона, Чебышёва на базе HDC требует создания специализированного типа данных, обеспечивающего автоматическое вычисление производных первого и второго порядков. В рамках данной работы на языке C# была разработана структура **Cdn** (см. Приложение1), включающая перегруженные арифметические операции и функции элементарной математики для HDC. А также создан класс **FractalCrypto**, содержащий процедуру **GenerateFractalKey(...)**, вычисляющую графический образ фрактала по заданной функции на заданной области комплексной плоскости. Эта процедура содержит три вложенных цикла анализирующих каждую точку исследуемой комплексной плоскости для выявления корней исходной функции и соответствующих областей притяжения. Перебор всех расчётных точек весьма ресурсоемкий, поэтому используются параллельные вычисления. Для вызова **GenerateFractalKey(...)** в список входных параметров входит указатель на исследуемую функцию, которая должна быть описана в терминах **Cdn.**

8. Экспериментальные исследования

8.1 Набор данных

Алгоритм был протестирован на наборе цветных RGB-изображений различного содержания и разрешения.

Исследование проводилось для функций:

1. $f_1(z) = z^3 - 1$
2. $f_2(z) = z^3 - z$
3. $f_3(z) = sin(z)$
4. $f_4(z) = sin(\cos(z))$
5. $f_5(z) = sin(\cos(\tan(z)))$

Для каждого изображения выполнялись:

1. шифрование;
2. построение гистограмм;
3. вычисление NPCR;
4. дешифрование.

Графические результаты некоторых тестируемых примеров представлены в Приложении 2.

Во всех экспериментах наблюдалось полное восстановление исходных данных.

8.2 Анализ гистограмм

Для оценки статистических характеристик были построены гистограммы каналов RGB до и после шифрования.

Для исходных изображений характерны выраженные пики распределений, связанные с содержимым сцены. После применения предложенного алгоритма распределения интенсивностей становятся весьма близкими к равномерным.

Полученные результаты демонстрируют эффективное разрушение статистических зависимостей и затрудняют применение методов частотного анализа.

8.3 Анализ NPCR

Для оценки чувствительности системы использовалась метрика Number of Pixel Change Rate (NPCR):

$$\mathrm{NPCR} = \frac{\sum_{i=1}^{w}\sum_{j=1}^{h} D(i,j)}{w \times h} \times 100\%,$$

где: w и h $-$ширина и высота изображения; $D(i,j) = 0$, если зашифрованные пиксели совпадают $(C_1(i,j) = C_2(i,j))$ или $D(i,j) = 1$, если они различаются $(C_1(i,j) \neq C_2(i,j))$.
Здесь сравниваются два шифр-изображения, полученных после изменения одного пикселя исходного изображения.

Для тестируемых примеров, если оригиналы отличаются на один пиксель, то в среднем NPCR ≈ 0.0003 – 0.00035%, а для зашифрованных изображений NPCR ≈ 98.3 – 99.5%.

Полученные значения NPCR близки к теоретически ожидаемым для криптографически стойких систем шифрования изображений, что подтверждает наличие выраженного лавинного эффекта.

9. Криптографический анализ

Предлагаемый алгоритм обладает следующими свойствами:

- высокой чувствительностью к параметрам ключа;
- сложным многомерным фрактальным ключевым пространством;

- разрушением корреляции между соседними пикселями;
- близкими к равномерным гистограммами шифртекстов;
- высокой чувствительностью к изменениям исходного изображения.

Дополнительным преимуществом является использование автоматического дифференцирования, позволяющего строить фракталы Ньютона и Чебышёва без ошибок численного дифференцирования. Ниже приводятся значения некоторых дополнительно проверенных характеристик криптостойкости для тестируемых примеров.

- UACI (Unified Average Changing Intensity — унифицированная средняя интенсивность изменений) для оценки стойкости алгоритмов шифрования изображений к дифференциальным атакам. Она показывает среднюю разность интенсивности пикселей между двумя зашифрованными изображениями, исходные файлы которых отличались всего на один пиксель. UACI ≈ 33.46%, что соответствует хаотическим системам.
- Для расчета информационной энтропии H использовалась формула Шеннона, которая измеряет степень неопределенности или случайности ключей и сообщений. H(cipher3) ≈ 7.999−8.000.

10. Заключение

Предложен новый алгоритм шифрования цветных изображений, основанный на применении фракталов Ньютона и Чебышёва в пространстве комплексных гипер-дуальных чисел. Комплексные гипер-дуальные числа используются не только для формирования высокоразмерной фрактальной динамики, но и как инструмент автоматического дифференцирования для вычисления первых и вторых производных.

Экспериментальные результаты, включающие анализ гистограмм и исследование NPCR, подтверждают высокую эффективность предложенного подхода. Полученные результаты демонстрируют перспективность использования фракталов Ньютона и Чебышёва на основе комплексных гипер-дуальных чисел при построении криптографических систем защиты графической информации.

Приложение 1

Полный C# код, используемый в численных экспериментах.

```
using System;
using System.Numerics;
using System.Drawing;
using System.Drawing.Imaging;
using System.Linq;
```

```
using System.Threading.Tasks;

// ---------Abridged version of HyperDualComplex (Cdn)---------
public struct HyperDualComplex
    {   public Complex x;   // f(z)
        public Complex e1;  // f'(z)
        public Complex e2;  // f''(z)
        public HyperDualComplex(Complex x, Complex e1, Complex e2)
        {   this.x = x;
            this.e1 = e1;
            this.e2 = e2;
        }
        public static HyperDualComplex operator +(HyperDualComplex a, HyperDualComplex b)
        { return new HyperDualComplex(a.x + b.x, a.e1 + b.e1, a.e2 + b.e2); }
        public static HyperDualComplex operator -(HyperDualComplex a, HyperDualComplex b)
        { return new HyperDualComplex(a.x - b.x, a.e1 - b.e1, a.e2 - b.e2); }
        public static HyperDualComplex operator *(HyperDualComplex a, HyperDualComplex b)
        { return new HyperDualComplex(
                    a.x * b.x,
                    a.x * b.e1 + b.x * a.e1,
                    a.x * b.e2 + b.x * a.e2 + 2.0 * a.e1 * b.e1
                );
        }
        public static HyperDualComplex operator /(HyperDualComplex a,
                                    HyperDualComplex b) => a * Inverse(b);
        public static HyperDualComplex Inverse(HyperDualComplex Z)
        {   Complex x0 = Z.x, x1 = Z.e1, x2 = Z.e2,
            x0_2 = x0 * x0, x0_3 = x0_2 * x0,
            y0 = Complex.One / x0, y1 = -x1 / x0_2,
            y2 = (2.0*x1*x1 - x0*x2) / x0_3;
            return new HyperDualComplex(y0, y1, y2);
        }
        public static HyperDualComplex Pow(HyperDualComplex a, int n)
        {HyperDualComplex r = new HyperDualComplex(Complex.One, Complex.Zero,
                                                                Complex.Zero);
            for (int i = 0; i < n; i++)
                r = r * a;
            return r;
        }
        public static HyperDualComplex Sin(HyperDualComplex Z)
        {   Complex x = Z.x, s = Complex.Sin(x), c = Complex.Cos(x),
                    c1 = c*Z.e1, c2 = c*Z.e2 - s * Z.e1* Z.e1;
            return new HyperDualComplex( s, c1, c2) ;
        }
        public static HyperDualComplex Cos(HyperDualComplex Z)
        {   Complex x = Z.x, s = Complex.Sin(x), c = Complex.Cos(x),
                    c1 = -s * Z.e1, c2 = -s * Z.e2 - c * Z.e1 * Z.e1;
            return new HyperDualComplex(s, c1, c2);
        }
        //--------------------- Functions under test---------------------
        public static HyperDualComplex F(HyperDualComplex Z) //Z^3 -1
        {   HyperDualComplex Z3 = HyperDualComplex.Pow(Z, 3);
            return new HyperDualComplex( Z3.x - Complex.One, Z3.e1, Z3.e2);
        }
        public static HyperDualComplex F1(HyperDualComplex Z) //Z^3 -Z
        {   HyperDualComplex Z3 = HyperDualComplex.Pow(Z, 3);
            return new HyperDualComplex(Z3.x - Z.x, Z.e1, Z.e2);
```

```
    }
    public static HyperDualComplex F2(HyperDualComplex Z) //Sin(Z)
    { return  HyperDualComplex.Sin(Z); }
    public static HyperDualComplex F3(HyperDualComplex Z) //Sin(Cos(Z))
    { return HyperDualComplex.Sin(Cos(Z)); }
    public static HyperDualComplex F4(HyperDualComplex Z) //Sin(Cos(Sin(Z)/Cos(Z)))
    { return HyperDualComplex.Sin(Cos(Sin(Z)/Cos(Z))); }
}

public class FractalCrypto
{   private readonly PixelFormat _pf = PixelFormat.Format32bppArgb;

   // ---------Cipher1---------
    public Bitmap EncodeCipher1(Bitmap src, int seed)
    {   int w = src.Width,  h = src.Height;
        Bitmap result = new Bitmap(w, h, _pf);
        Rectangle rect = new Rectangle(0, 0, w, h);
        BitmapData sData = src.LockBits(rect, ImageLockMode.ReadOnly, _pf);
        BitmapData rData = result.LockBits(rect, ImageLockMode.WriteOnly, _pf);
        int stride = sData.Stride;
        Random rng = new Random(seed);
        unsafe
        {   byte* sPtr = (byte*)sData.Scan0;
            byte* rPtr = (byte*)rData.Scan0;
            for (int y = 0; y < h; y++)
            {   byte* sRow = sPtr + y * stride;
                byte* rRow = rPtr + y * stride;
                for (int x = 0; x < w; x++)
                {   byte* pSrc = sRow + x * 4;
                    byte* pDst = rRow + x * 4;
                    byte kB = (byte)rng.Next(256);
                    byte kG = (byte)rng.Next(256);
                    byte kR = (byte)rng.Next(256);
                    byte kA = (byte)rng.Next(256);
                    pDst[0] = (byte)(pSrc[0] ^ kB);
                    pDst[1] = (byte)(pSrc[1] ^ kG);
                    pDst[2] = (byte)(pSrc[2] ^ kR);
                    pDst[3] = (byte)(pSrc[3] ^ kA);
                }
            }
        }
        src.UnlockBits(sData);
        result.UnlockBits(rData);
        return result;
    }
   public Bitmap DecodeCipher1(Bitmap cipher, int seed)
    { return EncodeCipher1(cipher, seed); }

   // ---------Cipher2---------
   public Bitmap EncodeCipher2(Bitmap src, byte[,] table)
    {   int w = src.Width,  h = src.Height;
        Bitmap result = new Bitmap(w, h, _pf);
        Rectangle rect = new Rectangle(0, 0, w, h);
        BitmapData sData = src.LockBits(rect, ImageLockMode.ReadOnly, _pf);
        BitmapData rData = result.LockBits(rect, ImageLockMode.WriteOnly, _pf);
        int stride = sData.Stride;
```

```
        unsafe
        {   byte* sPtr = (byte*)sData.Scan0;
            byte* rPtr = (byte*)rData.Scan0;
            for (int y = 0; y < h; y++)
            {   byte* sRow = sPtr + y * stride;
                byte* rRow = rPtr + y * stride;
                for (int x = 0; x < w; x++)
                {   byte* p = sRow + x * 4;
                    for (int c = 0; c < 4; c++)
                    {   byte P = p[c];
                        byte a = (byte)((P >> 4) & 0x0F);
                        byte b = (byte)(P & 0x0F);
                        byte newLow = table[a, b];
                        byte newP = (byte)((a << 4) | newLow);

                        rRow[x * 4 + c] = newP;
                    }
                }
            }
        }
        src.UnlockBits(sData);
        result.UnlockBits(rData);
        return result;
    }
    public Bitmap DecodeCipher2(Bitmap src, byte[,] invTable)
    {   int w = src.Width,  h = src.Height;
        Bitmap result = new Bitmap(w, h, _pf);
        Rectangle rect = new Rectangle(0, 0, w, h);
        BitmapData sData = src.LockBits(rect, ImageLockMode.ReadOnly, _pf);
        BitmapData rData = result.LockBits(rect, ImageLockMode.WriteOnly, _pf);
        int stride = sData.Stride;
        unsafe
        {   byte* sPtr = (byte*)sData.Scan0;
            byte* rPtr = (byte*)rData.Scan0;
            for (int y = 0; y < h; y++)
            {   byte* sRow = sPtr + y * stride;
                byte* rRow = rPtr + y * stride;
                for (int x = 0; x < w; x++)
                {   byte* p = sRow + x * 4;
                    for (int c = 0; c < 4; c++)
                    {   byte P = p[c];
                        byte a = (byte)((P >> 4) & 0x0F);
                        byte newLow = (byte)(P & 0x0F);
                        byte originalLow = invTable[a, newLow];
                        byte newP = (byte)((a << 4) | originalLow);
                        rRow[x * 4 + c] = newP;
                    }
                }
            }
        }
        src.UnlockBits(sData);
        result.UnlockBits(rData);
        return result;
    }
    // ---------Cipher3---------
    public Bitmap EncodeCipher3(Bitmap cipher2, Bitmap fractalKey)
    {   int w = cipher2.Width, h = cipher2.Height;
```

```
        Bitmap result = new Bitmap(w, h, _pf);
        Rectangle rect = new Rectangle(0, 0, w, h);
        BitmapData cData = cipher2.LockBits(rect, ImageLockMode.ReadOnly, _pf);
        BitmapData fData = fractalKey.LockBits(rect, ImageLockMode.ReadOnly, _pf);
        BitmapData rData = result.LockBits(rect, ImageLockMode.WriteOnly, _pf);
        int stride = cData.Stride;
        unsafe
        {   byte* cPtr = (byte*)cData.Scan0;
            byte* fPtr = (byte*)fData.Scan0;
            byte* rPtr = (byte*)rData.Scan0;
            for (int y = 0; y < h; y++)
            {   byte* cRow = cPtr + y * stride;
                byte* fRow = fPtr + y * stride;
                byte* rRow = rPtr + y * stride;
                for (int x = 0; x < w; x++)
                {  for (int c = 0; c < 4; c++)
                   {rRow[x * 4 + c] = (byte)(cRow[x * 4 + c] ^ fRow[x * 4 + c]);}
                }
            }
        }
        cipher2.UnlockBits(cData);
        fractalKey.UnlockBits(fData);
        result.UnlockBits(rData);
        return result;
    }
    public Bitmap DecodeCipher3(Bitmap cipher3, Bitmap fractalKey)
    { return EncodeCipher3(cipher3, fractalKey); }

    public Bitmap GenerateFractalKey(int width, int height,
                    double centerX, double centerY,
                    int maxIter, Func<HyperDualComplex, HyperDualComplex> F)
    {   double scale = 4.0 / Math.Min(width, height);
        double escapeRadius = 10.0;
        Bitmap bmp = new Bitmap((int)width, height, _pf);
        int[] pixels = new int[width * height];
        Parallel.For(0, height, py =>
        {   for (int px = 0; px < width; px++)
            {   double x = centerX + (px - width / 2.0) * scale;
                double y = centerY + (py - height / 2.0) * scale;
                Complex z = new Complex(x, y);
                int iter = 0;
                while (iter < maxIter)
                { HyperDualComplex Z = new HyperDualComplex(z, Complex.One,
                                                            Complex.Zero);
                    HyperDualComplex FZ = F(Z);
                    Complex f = FZ.x;
                    Complex fp = FZ.e1;
                    Complex fpp = FZ.e2;
                    if (fp.Magnitude < 1e-14) break;
                    Complex K = f / fp;
                    Complex L = fpp / fp;
                    z = z - K * (1.0 + 0.5 * K * L);
                    if (z.Magnitude > escapeRadius)  break;
                    iter++;
                }
                int R = (iter * 7) % 255;
                int G = (iter * 13) % 255;
```

```
                int B = (iter * 23) % 255;
                int argb = (255 << 24) | (R << 16) | (G << 8) | B;
                pixels[py * width + px] = argb;
            }
        });
        unsafe
        {   BitmapData data = bmp.LockBits(
                new Rectangle(0, 0, width, height),
                ImageLockMode.WriteOnly, _pf);
            byte* dst = (byte*)data.Scan0;
            int stride = data.Stride;
            for (int y = 0; y < height; y++)
            {   byte* row = dst + y * stride;
                int rowIndex = y * width;
                for (int x = 0; x < width; x++)
                {  int argb = pixels[rowIndex + x];
                    row[x * 4 + 0] = (byte)(argb & 0xFF);
                    row[x * 4 + 1] = (byte)((argb >> 8) & 0xFF);
                    row[x * 4 + 2] = (byte)((argb >> 16) & 0xFF);
                    row[x * 4 + 3] = (byte)((argb >> 24) & 0xFF);
                }
            }
            bmp.UnlockBits(data);
        }
        return bmp;
    }
    public  byte[,] GenerateCryptoTable16x16(int seed)
    {   byte[,] table = new byte[16, 16];
        Random rng = new Random(seed);
        for (int row = 0; row < 16; row++)
        {   byte[] perm = Enumerable.Range(0, 16).Select(i => (byte)i).ToArray();
            for (int i = 15; i > 0; i--)
            {   int j = rng.Next(i + 1);
                (perm[i], perm[j]) = (perm[j], perm[i]);
            }
            for (int col = 0; col < 16; col++) table[row, col] = perm[col];
        }
        return table;
    }

    public static byte[,] BuildInverseTable(byte[,] table)
    {   byte[,] inv = new byte[16, 16];
        for (int a = 0; a < 16; a++)
        {   for (int b = 0; b < 16; b++)
            {   byte mapped = table[a, b];
                inv[a, mapped] = (byte)b;
            }
        }
        return inv;
    }

}
     // ---------CALCULATION EXAMPLE---------
      Bitmap _original, _cipher1, _cipher2, _cipher3, _fractalKey;
      int _maxIter = 25;
      Func<HyperDualComplex, HyperDualComplex> _F = HyperDualComplex.F1;
```

```
// ---------Encode---------
 _original= new Bitmap(@"C:\path\to\your\image.bmp"));
 _seed = 10;// _seed = RandomNumberGenerator.GetInt32(1, 101);
 FractalCrypto _fr = new FractalCrypto();
 _cipher1 = _fr.EncodeCipher1(_original, _seed);
 _table = _fr.GenerateCryptoTable16x16(_seed);
 _cipher2 = _fr.EncodeCipher2(_cipher1, _table);
 int w = _cipher2.Width, h = _cipher2.Height;//2
 double scale = 4.0 / Math.Min(w, h);
 _fractalKey = _fr.GenerateFractalKey(w, h, 0, 0, _maxIter, _F);
 _cipher3 = _fr.EncodeCipher3(_cipher2, _fractalKey);

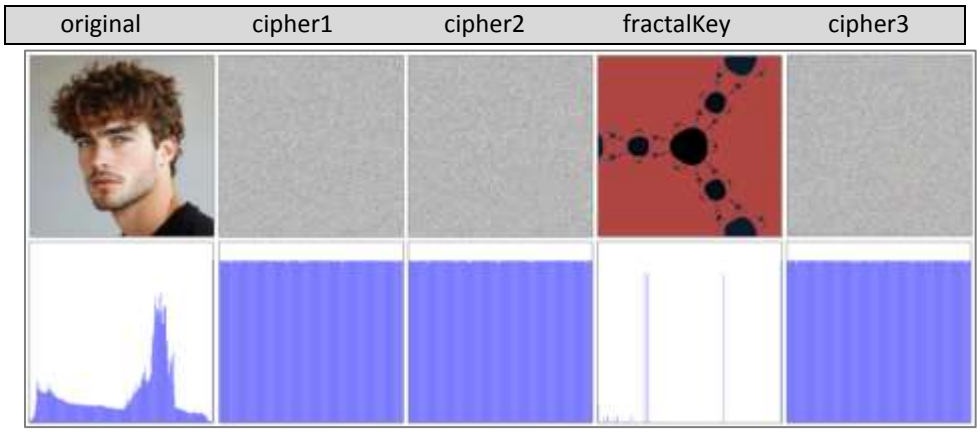
 // ---------Decode---------
 FractalCrypto _fr = new FractalCrypto();
 _table = _fr.GenerateCryptoTable16x16(_seed);
 var invTable = _fr.BuildInverseTable(_table)
 _fractalKey = _fr.GenerateFractalKey(_cipher3.Width,
                              _cipher3.Height, 0, 0, _maxIter, _F);
 _cipher2 = _fr.DecodeCipher3(_cipher3, _fractalKey);
 _cipher1 = _fr.DecodeCipher2(cipher2, invTable);
 _original = _fr.DecodeCipher1(cipher1, _seed);
```

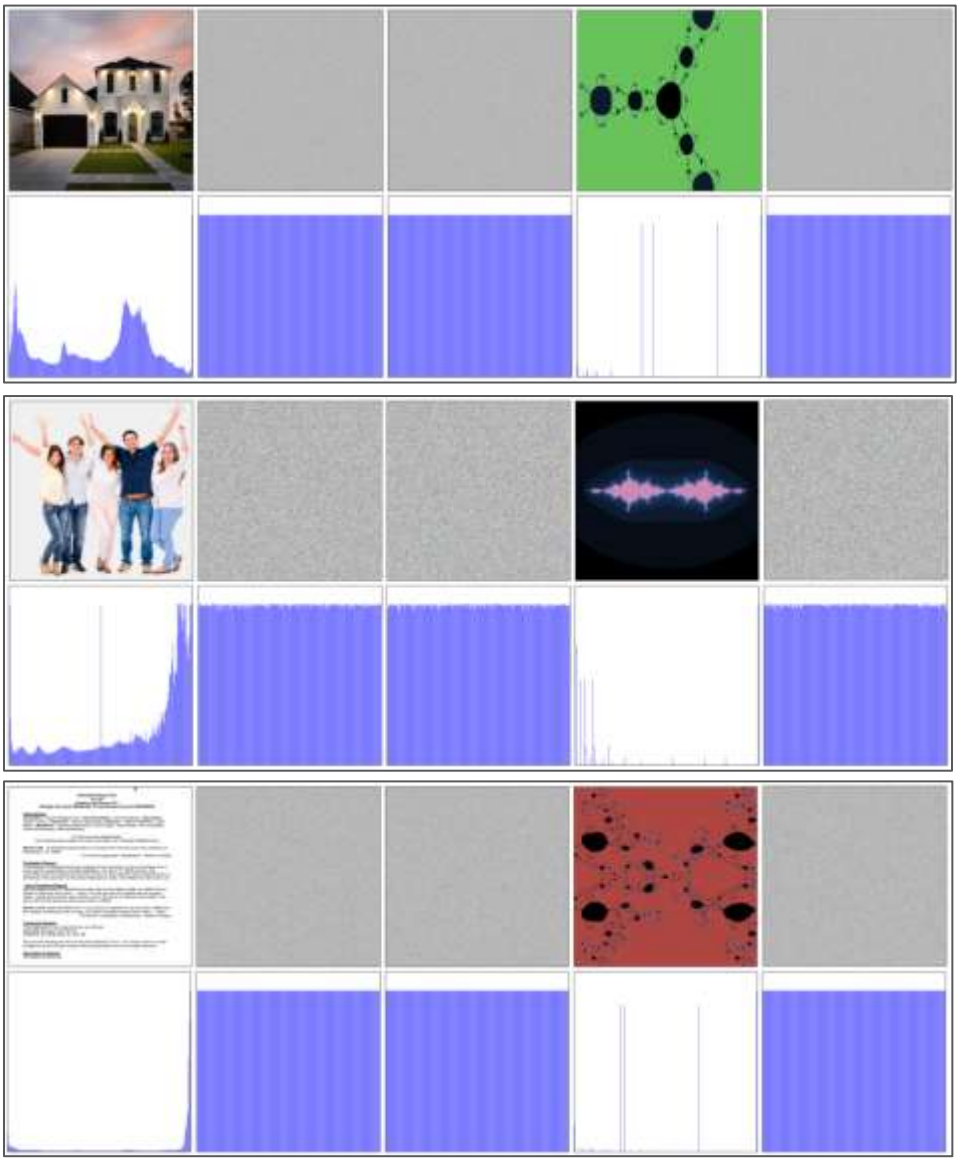
Приложение 2

Ниже представлены результаты некоторых тестовых примеров графической криптографии, выполненных на основе C# кода Приложения 1. Каждому примеру соответствуют две строки приведенной таблицы:

1) последовательность результатов шагов криптографического процесса;

2) гистограммы результатов каждого шага;





ЛИТЕРАТУРА

1. *Burton A.* Newton's method and fractals. Whitman College. – 2009. URL: https://www.whitman.edu/Documents/Academics/Mathematics/burton.pdf

2. *Nayak, T., Pal, S.* Symmetry and Dynamics of Chebyshev's Method. *Mediterr. J. Math.* **22**, 12 (2025).

3. *Олифер В. И.* Численное исследование динамики итерационных методов Ньютона, Чебышева и Хэлли на комплексной плоскости с использованием гипер-дуальных чисел.– 2026. URL:http://viosolutions.amerihomesrealty.com/pdf/Fractals_z.pdf

4. *Олифер В.И.* Усеченные гипер-дуальные числа в автоматическом дифференцировании.

– 2020.URL:http://viosolutions.amerihomesrealty.com/pdf/Усеченные_гипер-дуальные_числа_в_автоматическом_дифференцировании.pdf

5. *Fike J.A., Alonso J.J.* The Development of Hyper-Dual Numbers for Exact Second-Derivative Calculations. AIAA Aerospace Sciences Meeting, 2011.

Абстракт

В работе предложен алгоритм шифрования цветных изображений, основанный на использовании фракталов Ньютона и Чебышёва, построенных в пространстве комплексных гипер-дуальных чисел. Приведен код программного обеспечения и рассмотрены численные примеры, полученные на его основе.

This paper proposes an algorithm for the encryption of color images, based on the use of Newton and Chebyshev fractals constructed within the space of complex hyper-dual numbers. The software code is provided, and numerical examples generated using this approach are examined.

Ключевые слова *шифрования цветных изображений, комплексные уравнения, гипре-дуальные числа над полем комплексных чисел, методы Ньютона, Чебышева, автоматическое дифференцирование, фракталы, algorithm for the encryption of color images, complex equations, hyper-dual numbers over the field of complex numbers, Newton's, Chebyshev's methods, automatic differentiation, fractals.*

20 октября 2026 г