Izveštaj za projekat iz predmeta Zaštita informacija

Student : Andrija Cenić 18042

Sadržaj

Uvod

Implementacija algoritma RC6, Bifid cipher, Knapsack cipher, Mod kodera CTR primenjen na RC6 algoritam i TigerHash algoritam. Ucitavanje i upis u fajl, ucitavanje i upis u BMP fajl. Paralelizija upisa, čitanja i kriptovanja. Implementacija je u programskom jeziku C# (.NET 7)

RC6

[1]Algoritam RC6 je nadogradnja algoritma RC5. Ovaj algoritam šifrira podatke blokovski. Specificiran je kao RC6-w/r/b algoritam, gde w - veličina reči, r - broj rundi za šifrovanje i b - dužina ključa za enkripciju.
Ova implementacija koristi reči od 32b i 20 rundi za šifrovanje.

[1] Ronald L. Rivest & M. J. B. Robshaw & R. Sidnez & Y.L. Yin (1998) The RC6 Block Cipher, M.I.T. Laboratory for Computer Science, 545 Technology Square, Cambridge, RSA Laboratories.

Algoritam za distribuciju ključeva

```
1 reference
public void SetKey(string key)
{
    uint w_bytes = (uint)Math.Ceiling((float)W / 8);
    uint c = (uint)Math.Ceiling((float)key.Length / w_bytes);

    L = new uint[c];

    char[] keychar = key.ToCharArray();
    char[] normkey = new char[c * 4];
    Buffer.BlockCopy(keychar, 0, normkey, 0, keychar.Length * sizeof(char));
    Buffer.BlockCopy(normkey, 0, L, 0, (int)(c * 4));

    S[0] = Pw;
    for (int _i = 1; _i < 2 * R + 4; _i++)
    {
        S[_i] = (uint)((S[_i - 1] + Qw) % mod);
    }

    uint A = 0, B = 0, i = 0, j = 0;
    uint v = 3 * uint.Max(c, 2 * R + 4);
    for (uint s = 1; s <= v; s++)
    {
        A = S[i] = ROL((uint)((S[i] + A + B) % mod), 3);
        B = L[j] = ROL((uint)((L[j] + A + B) % mod), (int)(A + B));
        i = (i + 1) % (2 * R + 4);
        j = (j + 1) % c;
    }

}
```

[2]

Kriptovanje

Za kriptovanje se koriste 4 registra **A**, **B**, **C** i **D** i predhodno generisani ključ **S**. Predstavljen algoritam

[2] Paje, Rommel Evan & Sison, Ariel & Medina, Ruji. (2019). Multidimensional key RC6 algorithm. 33-38. 10.1145/3309074.3309095. link

Encryption with RC6-$w/r/b$

Input: Plaintext stored in four w-bit input registers A, B, C, D
Number r of rounds
w-bit round keys $S[0, \ldots, 2r + 3]$

Output: Ciphertext stored in A, B, C, D

Procedure:
$B = B + S[0]$
$D = D + S[1]$
for $i = 1$ **to** r **do**
{
$t = (B \times (2B + 1)) \lll \lg w$
$u = (D \times (2D + 1)) \lll \lg w$
$A = ((A \oplus t) \lll u) + S[2i]$
$C = ((C \oplus u) \lll t) + S[2i + 1]$
$(A, B, C, D) = (B, C, D, A)$
}
$A = A + S[2r + 2]$
$C = C + S[2r + 3]$

Implementacija

```
public uint[] Encrypt4Regs(uint[] data)
{
    uint[] encrypted = new uint[4];
    UInt32 A = data[0], B = data[1], C = data[2], D = data[3];

    B += S[0];
    D += S[1];

    uint t, u, temp;

    for (int i = 1; i <= R; i++)
    {
        t = ROL((uint)((B * (2 * B + 1)) % mod), (int)LOG_W);
        u = ROL((uint)((D * (2 * D + 1)) % mod), (int)LOG_W);
        A = ROL(A ^ t, (int)u) + S[2 * i];
        C = ROL(C ^ u, (int)t) + S[2 * i + 1];
        temp = A;
        A = B;
        B = C;
        C = D;
        D = temp;
    }
    A += S[2 * R + 2];
    C += S[2 * R + 3];

    encrypted[0] = A;
    encrypted[1] = B;
    encrypted[2] = C;
    encrypted[3] = D;

    return encrypted;
}
```

Dekriptovanje

Dekriptovanje je obrnuti proces od kriptovanja

Decryption with RC6-$w/r/b$

Input: Ciphertext stored in four w-bit input registers A, B, C, D
Number r of rounds
w-bit round keys $S[0, \ldots, 2r+3]$

Output: Plaintext stored in A, B, C, D

Procedure:
$C = C - S[2r+3]$
$A = A - S[2r+2]$
for $i = r$ **downto** 1 **do**
{
$(A, B, C, D) = (D, A, B, C)$
$u = (D \times (2D+1)) \lll \lg w$
$t = (B \times (2B+1)) \lll \lg w$
$C = ((C - S[2i+1]) \ggg t) \oplus u$
$A = ((A - S[2i]) \ggg u) \oplus t$
}
$D = D - S[1]$
$B = B - S[0]$

Implementacija:

```
3 references
public uint[] Decrypt4Regs(uint[] data)
{
    uint[] dec = new uint[4];

    uint A = data[0], B = data[1], C = data[2], D = data[3];
    uint t, u, temp;

    C -= S[2 * R + 3];
    A -= S[2 * R + 2];
    for (int i = (int)R; i >= 1; i--)
    {
        temp = D;
        D = C;
        C = B;
        B = A;
        A = temp;

        u = ROL((uint)((D * (2 * D + 1)) % mod), (int)LOG_W);
        t = ROL((uint)((B * (2 * B + 1)) % mod), (int)LOG_W);
        C = ROR((uint)((C - S[2 * i + 1]) % mod), (int)t) ^ u;
        A = ROR((uint)((A - S[2 * i]) % mod), (int)u) ^ t;
    }
    D -= S[1];
    B -= S[0];

    dec[0] = A;
    dec[1] = B;
    dec[2] = C;
    dec[3] = D;
    return dec;
}
```

Ostalo

Implementirano kritptovanje niza bajtova, string-a, niza karaktera. Najbitnije je šiforvanje niza bajtova.

Bifid cipher

[3]Ovaj algoritam se koristi za krtiptovanje teksta. Moguće je kriptovati 25 karaktera, svi ostali se zanemaruju ili u u slučaku karaktera ‘j’ koji se zamenjuje sa ‘i’.
Ključ kod ovog algoritma je matrica karaktera 5x5.
Kod kodiranja svaki karakter teksta koji se kodira se predstalja brojem vrste i kolone u kojoj se taj karater nalazi u ključu. Vrsta se upisuje u jedan niz dok se kolona upisuje u drugi niz.
Pri kodiranju se ta dva niza nadovezuju jedan na drugi formirajuci novi niz brojeva. Iz tog niza se uzimaju redom po dva broja koja predstavljaju kod tj. vrstu i kolonu novog kodiranog karaktera. Ovako se transformise ceo niz brojeva u nove karaktere i to predstavlja kodirani tekst. Dekodiranje je obrnuti proces.

Šifrovanje

```
1 reference
public string Encrypt(string text)
{
    text = text.ToUpper();
    int[] up = new int[text.Length];
    int[] down = new int[text.Length];
    int size = 0;
    for (int i = 0; i < text.Length; i++)
    {
        char curr = text[i];
        if (curr < 'A' || curr > 'Z')
            continue;
        if (curr == 'J')
            curr = 'I';

        var pair = map[curr];
        up[size] = pair.Key;
        down[size] = pair.Value;
        size++;
    }
    int[] coords = new int[size * 2];
    Buffer.BlockCopy(up, 0, coords, 0, size * sizeof(int));
    Buffer.BlockCopy(down, 0, coords, size * sizeof(int), size * sizeof(int));
    string rez = "";
    for (int k = 0; k < size * 2; k += 2)
    {
        int i = coords[k], j = coords[k + 1];
        rez += keyMatrix[i, j];
    }
    return rez;
}
```

[3] Bifid Cipher, link

Dešifrovanje

```
public string Decrypt(string text)
{
    string rez = "";
    int[] coords = new int[text.Length * 2];
    int size = 0;
    for (int i = 0; i < text.Length; i++)
    {
        char curr = text[i];
        if (curr < 'A' || curr > 'Z')
            continue;
        var kyp = map[curr];
        coords[size++] = kyp.Key;
        coords[size++] = kyp.Value;
    }
    int[] up = new int[size / 2];
    int[] down = new int[size / 2];

    Buffer.BlockCopy(coords, 0, up, 0, size * sizeof(int) / 2);
    Buffer.BlockCopy(coords, size * sizeof(int) / 2, down, 0, size * sizeof(int) / 2);

    for (int k = 0; k < size / 2; k++)
    {
        rez += keyMatrix[up[k], down[k]];
    }
    return rez;
}
```

Knapsack cipher

[4] [5]Knapsack cipher je algoritam sa asimetričnim ključevima. Kriptovanje se vrši sa javnim ključem dok se dekriptovanje vrši sa privatnim ključem.
Kljulevi su niz brojeva. U ovoj implementaciji ključevi su nizovi od 8 brojeva (kodiranje jednog bajta). Privatni ključ je niz strogo rastućih brojeva gde svaki broj je veći od zbira svih predhodnih.
Pored javnog i privatnog ključa potrebni su brojevi *N*, *M*, i *N_1*. *M* je broj koji je veći od zbira svih projeva u privatnom ključu, *N* je manji broj od *M*, i ne poseduje zajednički faktor sa brojem *M*.
Broj *N_1* je broj za koj važi : $N * N^{-1} \% M = 1$

[4] Knapsack Encryption Algorithm in Cryptography, link
[5] Knapsack, str 68 - 69, Zaštita Informacija, Vladan Vučković & Petar Rajković

Generisanje javnog ključa

```
1 reference
public KnapsackCypher(int N, int M, int[] privateKey)
{
    this.N = N;
    this.M = M;
    this.privateKey = privateKey;

    publicKey = new int[8];
    GeneratePublicKey();
}

1 reference
private void GeneratePublicKey()
{
    for (int i = 0; i < 8; i++)
    {
        publicKey[i] = (privateKey[i] * N) % M;
        // Console.Write(publicKey[i] + " ");
    }
    // Console.WriteLine();
    N_1 = 1;
    while ((N * N_1) % M != 1) N_1++;
    // Console.WriteLine(N_1);
}
```

Kriptovanje

Za kriptovanje u ovom algoritmu koristi se vrednost svih bitova u bajtu. Suma proizvoda svakog bita sa odgavarajućim vrednostima iz javnog ključa je zapravo šifra tog bita.
javni ključ : 101 202 91 283 152 304 83 55
privatni ključ : 1 2 4 9 17 34 69 140
N : 101
M: 313:
N_1 : 31
bit za šifrovanje 10001000 => 1 * 101 + 0 * 202 + .. + 1* 152 + 0* 304 + … = 253

```
public int[] Encrypt(byte[] bytes)
{
    int index = 0;
    int[] rez = new int[bytes.Length];
    foreach (byte b in bytes)
    {
        int cr = 0;
        byte v = 1;
        for (int i = 7; i >= 0; i--)
        {
            if ((b & v) > 0)
            {
                cr += publicKey[i];
            }
            v = (byte)(v << 1);
        }
        rez[index++] = cr;
    }
    return rez;
}
```

Dekriptovanje

Dekriptovanje se vrši po formuli $X * N^{-1} \% M$. Dobijeni broj se kodira privatnim ključem. Prvo se gleda bit sa naivećom težinom.
Isti podaci kao za šifrovanje :
253 => 253 * 31 % 313 = 18
18 >= 140 **=>** 0
18 >= 69 **=>** 0
18 >= 34 **=>** 0
18 >= 17 **=>** 1
18-17 = 1 >= 9 **=>** 0
1 >= 4 **=>** 0
1 >= 2 **=>** 0
1 >= 1 **=>** 1
Kada lepo pročitamo broj => 1000100 -> broj koji smo šifrirali.

```
2 references
public byte[] Decrypt(int[] crpt)
{
    int index = 0;
    byte[] bytes = new byte[crpt.Length];
    foreach (int i in crpt)
    {
        int curr = (i * N_1) % M;
        int j = 7;
        byte v = 1;
        byte r = 0;
        while (curr > 0 && j >= 0)
        {
            if (curr >= privateKey[j])
            {
                curr -= privateKey[j];
                r = (byte)(r | v);
            }
            v = (byte)(v << 1);
            j--;
        }
        bytes[index++] = r;
    }
    return bytes;
}
```

CRT mod kodera primenjen na RC6

[6]Ovaj mod kodera omogućava da se kriptovanje i dekriptovanje kod blok kodera da se vrši nad nizom podataka. Poseduje *IV* (vektor inicijalizacije) i menja se kriptovanje i dekriptovanje, tj ne vrši se nad podacime već nad ključem pa se bitovi sabiraju po modulu 2.

$C_i = P_i \oplus E(IV + i, K), i = 0, 1, 2...$

$P_i = C_i \oplus E(IV + i, K), i = 0, 1, 2...$

[6] Knapsack, str 48, Zaštita Informacija, Vladan Vučković & Petar Rajković

Implementacija

```
private byte[] IV;
1 reference
public RC6CRT(string key, byte[]? iv = null) : base(key)
{
    if (iv != null && iv.Length == 4 * sizeof(uint))
        IV = iv;
    else
    {
        IV = new byte[4 * sizeof(uint)];
        IV[0] = 131;
        int i = 1;
        for (; i < 4 * sizeof(uint); i++)
        {
            IV[i] = (byte)(IV[i - 1] * 17 / 2);
        }
    }
}
```

```
public byte[] DecryptByteArrayCRT(byte[] input)
{
    byte[] dec = new byte[input.Length];
    uint[] data = new uint[4];

    uint[] currIV = new uint[4];
    Buffer.BlockCopy(IV, 0, currIV, 0, 4 * sizeof(uint));

    for (int i = 0; i < input.Length / (4 * sizeof(uint)); i++)
    {
        Buffer.BlockCopy(currIV, 0, data, 0, 4 * sizeof(uint));
        uint[] rez = Encrypt4Regs(data);
        Buffer.BlockCopy(input, i * 4 * sizeof(uint), data, 0, 4 * sizeof(uint));
        for (int j = 0; j < 4; j++)
        {
            rez[j] = rez[j] ^ data[j];
        }
        Buffer.BlockCopy(rez, 0, dec, i * 4 * sizeof(uint), 4 * sizeof(uint));
        for (int j = 0; j < 4; j++)
        {
            currIV[j] = (currIV[j] + ((uint)i + 1));
        }
    }

    return dec;
}
```

```
public byte[] EncryptByteArrayCRT(byte[] input)
{
    int missing = input.Length % (4 * sizeof(uint));
    missing = missing == 0 ? 0 : 4 * sizeof(uint) - missing;
    int len = input.Length + missing;

    byte[] inputExtended = new byte[len];
    Buffer.BlockCopy(input, 0, inputExtended, 0, input.Length);

    byte[] enc = new byte[len];

    uint[] data = new uint[4];

    uint[] currIV = new uint[4];
    Buffer.BlockCopy(IV, 0, currIV, 0, 4 * sizeof(uint));

    for (int i = 0; i < len / (4 * sizeof(uint)); i++)
    {
        Buffer.BlockCopy(currIV, 0, data, 0, 4 * sizeof(uint));
        uint[] rez = Encrypt4Regs(data);
        Buffer.BlockCopy(inputExtended, i * 4 * sizeof(uint), data, 0, 4 * sizeof(uint));
        for (int j = 0; j < 4; j++)
        {
            rez[j] = rez[j] ^ data[j];
        }
        Buffer.BlockCopy(rez, 0, enc, i * 4 * sizeof(uint), 4 * sizeof(uint));
        for (int j = 0; j < 4; j++)
        {
            currIV[j] = (currIV[j] + ((uint)i + 1));
        }
    }

    return enc;
}
```

Upis i čitanje iz fajla

Kreirana klasa Encryption koja sadrži metode za čitanje i upis u fajl primenom nekog od algoritama. Algoritmima se pristupa preko interfejsa AlgorithmInteface :

```
6 references
interface AlgorithmInterface
{
    4 references
    public byte[] Encrypt(byte[] input);
    4 references
    public byte[] Decrypt(byte[] input);
}
```

Za sve algoritme definition interfejs, primer RC6CRT :

```
namespace Algorithms.Interfaces
{
    0 references
    class RC6CRTInterface : AlgorithmInterface
    {
        3 references
        private RC6CRT algorithm;
        0 references
        public RC6CRTInterface(string key, byte[]? iv = null)
        {
            algorithm = new RC6CRT(key, iv);
        }
        4 references
        public byte[] Encrypt(byte[] input)
        {
            return algorithm.EncryptByteArrayCRT(input);
        }
        4 references
        public byte[] Decrypt(byte[] input)
        {
            return algorithm.DecryptByteArrayCRT(input);
        }
    }
}
```

Čitanje i upis u bilo koji fajl

```
0 references
public void EncryptFile(string inputName, string outputName)
{
    FileStream input = new FileStream(inputName, FileMode.Open, FileAccess.Read);
    FileStream output = new FileStream(outputName, FileMode.Create, FileAccess.Write);
    byte[] data = new byte[512];
    while (input.Read(data, 0, 512) > 0)
    {
        byte[] enc = Algorithm.Encrypt(data);
        output.Write(enc);
    }
    output.Close();
    input.Close();
}
```

```
0 references
public void DecryptFile(string inputName, string outputName)
{
    FileStream input = new FileStream(inputName, FileMode.Open, FileAccess.Read);
    FileStream output = new FileStream(outputName, FileMode.Create, FileAccess.Write);
    byte[] data = new byte[512];
    while (input.Read(data, 0, 512) > 0)
    {
        byte[] enc = Algorithm.Decrypt(data);
        output.Write(enc);
    }
    output.Close();
    input.Close();
}
```

Primenljivo na bilo koji od algoritama, osim za bifid ako se ne radi o tekstu.

Čitanje i upis u BMP fajl

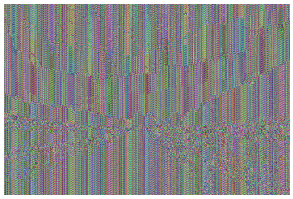
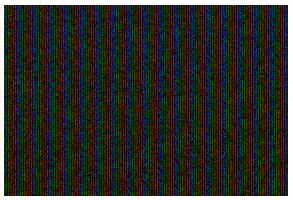
```
0 references
public void EncryptBMPFile(string inputName, string outputName)
{
    FileStream input = new FileStream(inputName, FileMode.Open, FileAccess.Read);
    byte[] header = new byte[54];
    int count = input.Read(header, 0, 54);
    FileStream output = new FileStream(outputName, FileMode.Create, FileAccess.Write);
    output.Write(header, 0, 54);

    byte[] data = new byte[512];
    while (input.Read(data, 0, 512) > 0)
    {
        byte[] enc = Algorithm.Encrypt(data);
        output.Write(enc);
    }

    output.Close();
    input.Close();
}
```

```
0 references
public void DecryptBMPFile(string inputName, string outputName)
{
    FileStream input = new FileStream(inputName, FileMode.Open, FileAccess.Read);
    byte[] header = new byte[54];
    int count = input.Read(header, 0, 54);
    FileStream output = new FileStream(outputName, FileMode.Create, FileAccess.Write);
    output.Write(header, 0, 54);

    byte[] data = new byte[512];
    while (input.Read(data, 0, 512) > 0)
    {
        byte[] enc = Algorithm.Decrypt(data);
        output.Write(enc);
    }

    output.Close();
    input.Close();
}
```

Primer za RC6 CRT algoritam:

```
Encryption file = new Encryption(new RC6CRTInterface("Ovo je kljuc za kriptovanje"));
file.EncryptBMPFile("bmp_24.bmp", "bmp_24_enc.bmp");
file.DecryptBMPFile("bmp_24_enc.bmp", "bmp_24_dec.bmp");
```

Rezultati:

	Originalna slika	Kriptovana	Dekriptovana
RC6 CRT			
RC6			
Knapsack			

Paralelizacija

Stari način

```
0 references
public byte[] EncryptParallel(byte[] input, int numOfThreads)
{
    int blockSize = input.Length / numOfThreads;
    byte[][] output = new byte[numOfThreads][];
    int fullLenght = 0;
    Parallel.For(0, numOfThreads, index =>
    {
        int bSize = blockSize;
        if (index == numOfThreads - 1 && input.Length > numOfThreads * blockSize)
            bSize += input.Length - numOfThreads * blockSize;
        byte[] curr = new byte[bSize];
        Buffer.BlockCopy(input, index * blockSize, curr, 0, bSize);
        byte[] enc = Algorithm.Encrypt(curr);
        output[index] = new byte[enc.Length];
        Interlocked.Add(ref fullLenght, enc.Length);
        Buffer.BlockCopy(enc, 0, output[index], 0, enc.Length);
    });
    byte[] finalOutput = new byte[fullLenght];
    int currentFill = 0;
    foreach (byte[] b in output)
    {
        Buffer.BlockCopy(b, 0, finalOutput, currentFill, b.Length);
        currentFill += b.Length;
    }
    return finalOutput;
}
```

```
0 references
public byte[] DecryptParallel(byte[] input, int numOfThreads, bool removePadding = false)
{
    //byte[] output = new byte[input.Length];
    byte[][] output = new byte[numOfThreads][];
    int fullLenght = 0;
    int blockSize = input.Length / numOfThreads;
    Parallel.For(0, numOfThreads, index =>
    {
        int bSize = blockSize;
        if (index == numOfThreads)
            bSize += input.Length - numOfThreads * blockSize;

        byte[] curr = new byte[bSize];
        Buffer.BlockCopy(input, index * blockSize, curr, 0, bSize);
        byte[] dec = Algorithm.Decrypt(curr);
        output[index] = new byte[dec.Length];
        Interlocked.Add(ref fullLenght, dec.Length);
        Buffer.BlockCopy(dec, 0, output[index], 0, dec.Length);
    });
    byte[] finalOutput = new byte[fullLenght];
    int currentFill = 0;
    foreach (byte[] b in output)
    {
        Buffer.BlockCopy(b, 0, finalOutput, currentFill, b.Length);
        currentFill += b.Length;
    }
    if (removePadding)
    {
        //Remove padding
        int numOfPadding = 0;
        for (int i = 0; i < output.Length; i += 2)
            if (finalOutput[i] == 0 & finalOutput[i + 1] == 0)
                numOfPadding++;
        byte[] finalOutput2 = new byte[output.Length - numOfPadding * 2];
        for (int i = 0, j = 0; i < output.Length; i += 2)
        {
            if (finalOutput[i] == 0 & finalOutput[i + 1] == 0)
                continue;
            finalOutput2[j] = finalOutput[i];
            finalOutput2[j + 1] = finalOutput[i + 1];
            j += 2;
        }
        return finalOutput2;
    }
    return finalOutput;
}
```

Pokazalo se da ne radi za za bilo koj broj thread-a.

Novi način

Razdvojio sam učitavanje i kriptovanje tako da nebi dolazilo do greške.

Učitavanje

```
public byte[] LoadFileParallel(string filePath, int numThreads)
{
    long fileSize = new FileInfo(filePath).Length;
    int chunkSize = (int)Math.Ceiling((double)fileSize / numThreads);
    byte[] file = new byte[chunkSize * numThreads];

    var chunks = Enumerable.Range(0, numThreads).Select(i =>
    {
        int start = i * chunkSize;
        return new { start, i };
    }).ToArray();

    var tasks = chunks.Select(chunk => Task.Run(() =>
    {
        var stream = new FileStream(filePath, FileMode.Open, FileAccess.Read);
        var buffer = new byte[chunkSize];
        stream.Seek(chunk.start, SeekOrigin.Begin);
        int bytesRead = stream.Read(buffer, 0, chunkSize);
        Buffer.BlockCopy(buffer, 0, file, chunk.start, bytesRead);
        stream.Close();
    })).ToArray();
    Task.WaitAll(tasks);
    return file;
}
```

Kriptovanje i Dekriptovanje za RC6 algoritam

Može se iz interfejsa pozivati *EncryptParallel* ili pozvati direktno iz algoritma

```
6 references
interface AlgorithmInterface
{
    4 references
    public byte[] Encrypt(byte[] input);
    4 references
    public byte[] Decrypt(byte[] input);

    0 references
    public byte[] EncryptParallel(byte[] input);
    0 references
    public byte[] DecryptParallel(byte[] input);
}
```

```
1 reference
public byte[] EncryptByteArrayParallel(byte[] input)
{
    int missing = input.Length % (4 * sizeof(uint));
    missing = missing == 0 ? 0 : 4 * sizeof(uint) - missing;
    int len = input.Length + missing;

    byte[] inputExtended = new byte[len];
    Buffer.BlockCopy(input, 0, inputExtended, 0, input.Length);

    byte[] enc = new byte[len];

    uint[] data = new uint[4];

    Parallel.For(0, len / (4 * sizeof(uint)), i =>
    {
        Buffer.BlockCopy(inputExtended, i * 4 * sizeof(uint), data, 0, 4 * sizeof(uint));
        uint[] rez = Encrypt4Regs(data);
        Buffer.BlockCopy(rez, 0, enc, i * 4 * sizeof(uint), 4 * sizeof(uint));
    });

    return enc;
}
```

```
public byte[] DecryptByteArrayParallel(byte[] input)
{
    byte[] dec = new byte[input.Length];
    uint[] data = new uint[4];
    Parallel.For(0, input.Length / (4 * sizeof(uint)), i =>
    {
        Buffer.BlockCopy(input, i * 4 * sizeof(uint), data, 0, 4 * sizeof(uint));
        uint[] rez = Decrypt4Regs(data);
        Buffer.BlockCopy(rez, 0, dec, i * 4 * sizeof(uint), 4 * sizeof(uint));
    });

    return dec;
}
```

Upis u fajl

```
2 references
public void SaveFile(byte[] file, string filePath)
{
    FileStream output = new FileStream(filePath, FileMode.Create, FileAccess.Write);
    output.Write(file, 0, file.Length);
    output.Close();
}
```

Nike moguće parallelizovati upis u fajl!

Korišćenje funkcionalnosti

```
1 reference
public void EncryptFileParallelNew(string inputName, string outputName, int numOfThreads)
{
    byte[] bytes = LoadFileParallel(inputName, numOfThreads);
    byte[] enc = Algorithm.EncryptParallel(bytes);
    SaveFile(enc, outputName);
}

1 reference
public void DecryptFileParallelNew(string inputName, string outputName, int numOfThreads)
{
    byte[] bytes = LoadFileParallel(inputName, numOfThreads);
    byte[] dec = Algorithm.DecryptParallel(bytes);
    SaveFile(bytes, outputName);
}
```

```
1 reference
public void EncryptFileRC6Parallel(string inputFile, string encFile = "enc.bin", string decFile = "dec.bin")
{
    Encryption e = new Encryption(new RC6Interface(KeyRC6));
    e.EncryptFileParallelNew(inputFile, encFile, 4);
    e.DecryptFileParallelNew(encFile, decFile, 4);
}
```

TigerHash

[7] [8] Tiger hash je 64-bit optimizovan algoritam čiji hash proizvodi 192bitni hash. Implementiran je uz pomoć HashAlgoritham klase koja već postoji u C# jeziku (.NET 7)

Distribucija ključeva

```
2 references
private void KeySchedule(ref ulong x0, ref ulong x1, ref ulong x2, ref ulong x3, ref ulong x4, ref ulong x5, ref ulong x6, ref ulong x7)
{
    x0 -= x7 ^ 0xA5A5A5A5A5A5A5A5UL;
    x1 ^= x0;
    x2 += x1;
    x3 -= x2 ^ ((~x1) << 19);
    x4 ^= x3;
    x5 += x4;
    x6 -= x5 ^ ((ulong)(~x4) >> 23);
    x7 ^= x6;
    x0 += x7;
    x1 -= x0 ^ ((~x7) << 19);
    x2 ^= x1;
    x3 += x2;
    x4 -= x3 ^ ((ulong)(~x2) >> 23);
    x5 ^= x4;
    x6 += x5;
    x7 -= x6 ^ 0x0123456789ABCDEFUL;
}
```

[7] Tigerhash, str 75-77, Zaštita Informacija, Vladan Vučković & Petar Rajković

[8] Tiger Hash implementation, caprnmac Chief Technology Officer Software Union link

Runda

```
24 references
private void Round(ref ulong x, ref ulong y, uint zh, uint zl)
{
    x -= t1[(int)(byte)zl] ^ t2[(int)(byte)(zl >> 16)] ^ t3[(int)(byte)zh] ^ t4[(int)(byte)(zh >> 16)];
    y += t4[(int)(byte)(zl >> 8)] ^ t3[(int)(byte)(zl >> 24)] ^ t2[(int)(byte)(zh >> 8)] ^ t1[(int)(byte)(zh >> 24)];
}
```

Hash-iranje

```
protected void ProcessBlock(byte[] inputBuffer, int inputOffset, int blockCount)
{
    ulong a = Accumulator[0], b = Accumulator[1], c = Accumulator[2], x0, x1, x2, x3, x4, x5, x6, x7;

    int i, spaceNeeded = blockCount * 8;
    if (x.Length < spaceNeeded) Array.Resize(ref x, spaceNeeded);

    Buffer.BlockCopy(inputBuffer, inputOffset, x, 0, blockCount * BlockSizeInput);

    for (i = -1; blockCount > 0; --blockCount, inputOffset += BlockSizeInput)
    {
        x0 = x[++i]; x1 = x[++i]; x2 = x[++i]; x3 = x[++i];
        x4 = x[++i]; x5 = x[++i]; x6 = x[++i]; x7 = x[++i];

        c ^= x0; Round(ref a, ref b, (uint)(c >> 32), (uint)c); b *= 5;
        a ^= x1; Round(ref b, ref c, (uint)(a >> 32), (uint)a); c *= 5;
        b ^= x2; Round(ref c, ref a, (uint)(b >> 32), (uint)b); a *= 5;
        c ^= x3; Round(ref a, ref b, (uint)(c >> 32), (uint)c); b *= 5;
        a ^= x4; Round(ref b, ref c, (uint)(a >> 32), (uint)a); c *= 5;
        b ^= x5; Round(ref c, ref a, (uint)(b >> 32), (uint)b); a *= 5;
        c ^= x6; Round(ref a, ref b, (uint)(c >> 32), (uint)c); b *= 5;
        a ^= x7; Round(ref b, ref c, (uint)(a >> 32), (uint)a); c *= 5;

        KeySchedule(ref x0, ref x1, ref x2, ref x3, ref x4, ref x5, ref x6, ref x7);

        b ^= x0; Round(ref c, ref a, (uint)(b >> 32), (uint)b); a *= 7;
        c ^= x1; Round(ref a, ref b, (uint)(c >> 32), (uint)c); b *= 7;
        a ^= x2; Round(ref b, ref c, (uint)(a >> 32), (uint)a); c *= 7;
        b ^= x3; Round(ref c, ref a, (uint)(b >> 32), (uint)b); a *= 7;
        c ^= x4; Round(ref a, ref b, (uint)(c >> 32), (uint)c); b *= 7;
        a ^= x5; Round(ref b, ref c, (uint)(a >> 32), (uint)a); c *= 7;
        b ^= x6; Round(ref c, ref a, (uint)(b >> 32), (uint)b); a *= 7;
        c ^= x7; Round(ref a, ref b, (uint)(c >> 32), (uint)c); b *= 7;

        KeySchedule(ref x0, ref x1, ref x2, ref x3, ref x4, ref x5, ref x6, ref x7);

        a ^= x0; Round(ref b, ref c, (uint)(a >> 32), (uint)a); c *= 9;
        b ^= x1; Round(ref c, ref a, (uint)(b >> 32), (uint)b); a *= 9;
        c ^= x2; Round(ref a, ref b, (uint)(c >> 32), (uint)c); b *= 9;
        a ^= x3; Round(ref b, ref c, (uint)(a >> 32), (uint)a); c *= 9;
        b ^= x4; Round(ref c, ref a, (uint)(b >> 32), (uint)b); a *= 9;
        c ^= x5; Round(ref a, ref b, (uint)(c >> 32), (uint)c); b *= 9;
        a ^= x6; Round(ref b, ref c, (uint)(a >> 32), (uint)a); c *= 9;
        b ^= x7; Round(ref c, ref a, (uint)(b >> 32), (uint)b); a *= 9;

        // feed forward
        a = Accumulator[0] ^= a; b -= Accumulator[1]; Accumulator[1] = b; c = Accumulator[2] += c;
    }
}
```

Servis za razmenu podataka

Servis za razmenu podataka može da posalje ključ za sve algoritme pomenute, kao i da kriptuje i vraća poruke, sa ključem koji je njemu poslat.

Za ključeve

```
app.MapGet("/KnapsackKey", () => new KnapsackKey { key = publicKey });
app.MapGet("/BifidKey", () => new BifidKey { key = bifidKey });
app.MapGet("/RC6Key", () => new RC6Key { key = rc6Key });
app.MapGet("/RC6CRTKey", () => new RC6Key { key = rc6crtKey });
```

TigerHash

```
app.MapPost("/tigerHash", (TigerHashData data) =>
{
    TigerHashData rsp = new TigerHashData();
    rsp.data = hash.ComputeHash(data.data!);
    return rsp;
});
```

Knapsack

```
app.MapPost("/knapsackSendEncrypted", (KnapsackDataEncrypted kdata) =>
{
    byte[] dec = knapsackCipher.Decrypt(kdata.data!);
    char[] carr = new char[dec.Length / 2];
    Buffer.BlockCopy(dec, 0, carr, 0, dec.Length);
    string s = new String(carr);
    Console.WriteLine("Klijent kaze :" + s);
    KnapsackDataEncrypted kde = new KnapsackDataEncrypted();
    string msg = "Primljeno! Pozdrav";
    byte[] msgByte = new byte[msg.Length * sizeof(char)];
    Buffer.BlockCopy(msg.ToCharArray(), 0, msgByte, 0, msgByte.Length);
    kde.data = KnapsackCypher.EncryptWithKey(msgByte, kdata.senderPublicKey!);
    return kde;

});
```

Bifid

```
app.MapPost("/bifidSendEncrypted", (BifidData bdata) =>
{
    string s = bifidCipher.Decrypt(bdata.data!);
    Console.WriteLine("Klijent kaze :" + s);
    Bifid clientBifid = new Bifid(bdata.senderKey!);
    string msg = clientBifid.Encrypt("Primljeno! Pozdrav!");
    BifidData msgData = new BifidData();
    msgData.data = msg;
    msgData.senderKey = bifidKey;
    return msgData;
});
```

RC6

```
app.MapPost("/RC6SendEncrypted", (RC6Data data) =>
{
    string s = rc.DecodeStringFaster(data.data!);
    Console.WriteLine("Klijent kaze : " + s);

    RC6 clientRC = new RC6(data.senderKey!);
    RC6Data ndata = new RC6Data();
    ndata.data = clientRC.EncryptStringFaster("Primljeno! Pozdrav!");
    ndata.senderKey = rc6Key;
    return ndata;
});
```

RC6 sa modom kodera CRT

```
app.MapPost("/RC6CRTSendEncrypted", (RC6CRTData data) =>
{
    byte[] dec = rc6crt.DecryptByteArrayCRT(data.data!);
    char[] charDec = new char[dec.Length / 2];
    Buffer.BlockCopy(dec, 0, charDec, 0, dec.Length);
    Console.WriteLine("Klijent kaze : " + (new String(charDec)));

    RC6CRT clientRc = new RC6CRT(data.senderKey!);
    string msg = "Primljeno! Pozdrav!";
    byte[] zaKodiranje = new byte[msg.Length / 2];
    Buffer.BlockCopy(msg.ToCharArray(), 0, zaKodiranje, 0, zaKodiranje.Length);

    RC6CRTData ndata = new RC6CRTData();
    ndata.senderKey = rc6crtKey;
    ndata.data = clientRc.EncryptByteArrayCRT(zaKodiranje);

    return ndata;
});
```

Klijent

Može da poziva sledeće komande :

- **knapsack** - Koristeći ovaj algoritam šalje poruku servisu
- **bifid** - Koristeći ovaj algoritam šalje poruku servisu
- **rc6** - Koristeći ovaj algoritam šalje poruku servisu
- **rc6crt** - Koristeći ovaj algoritam šalje poruku servisu
- **hash** - Salje niz bajtova servisu i on ih hesh-ira

- **file** <ime fajla> <ime algoritma> <ulazni fajl> <enkriptpvan fajl> <dek. fajl> - kriptuje fajl
- **filep** <ulazni fajl> <enkriptpvan fajl> <dek. fajl> - kriptuje fajl koristeći paralelizaciju
- **bmp** <ime fajla> <ime algoritma> <ulazni fajl> <enkriptpvan fajl> <dek. fajl> - kriptuje **bmp** fajl
- **end** - izlaz iz programa

Klijent koristi predhodno pomenute algoritme za realizaciju svih funkcionalnosti.