Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Исследование ассиметричных шифров**

Студент: Рубашек А. А.

ФИТ 3 курс 5 группа

Преподаватель:

Савельева Маргарита Геннадьевна

1. **Цель работы**

Изучение и приобретение практических навыков разработки и использования приложений для реализации блочных шифров.

1. **Задание**

Разработать авторские многооконные приложения в соответствии с целью лабораторной работы. При этом можно воспользоваться готовыми библиотеками либо программными кодами, реализующими заданные алгоритмы.

Приложение 1 должно реализовывать генерацию ПСП в соответствии с вариантом из табл. 6.6.

Приложение 2 должно реализовывать алгоритм RC4 в соответствии с вариантом из табл. 6.7, а также дополнительно выполнять оценку скорости выполнения операций генерации ПСП.

1. **Ход работы**

В соответствии с заданиями, необходимо было реализовать генерацию сверхвозрастающей последовательности (в простейшем случае принимается z = 6 (для кодировки Base64) и z = 8 (для кодировки ASCII)), вычисление нормальной последовательности (открытого ключа), зашифрование сообщения, состоящего из собственных фамилии, имени и отчества, расшифрование сообщения, оценку времени выполнения операций зашифрования и расшифрования. Для реализации всех этих функций был разработан класс RanceCipher, представленный на рисунке 3.1.

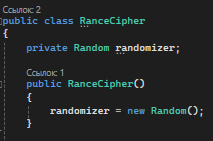


Рисунок 3.1 – Класс RanceCipher

В этом классе были реализованы функции для генерации сверхвозрастающей последовательности (закрытого ключа) и для генерации нормальной последовательности (открытого ключа). Эти функции представлены на в Приложении 1.

А для шифрования и расшифрования сообщения были разработаны функции Encrypt() и Decrypt(), представленные в Приложении 2.

В результате, при сгенерированных *d* и *e* с помощью разработанных выше функций, *M* = «Rubashek Alexandr Alexandrovich», *n* = 8, *a* = 33, получим следующий вывод, представленный на рисунке 3.2.

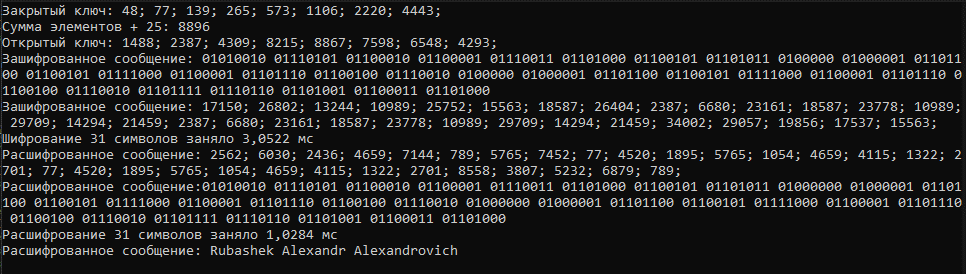


Рисунок 3.2 – Результат работы класса RanceCipher

Также, надо было оценить скорость выполнения операций шифрования и дешифрования. Результаты мы можем увидеть на рисунках 3.3 – 3.4

Рисунок 3.3 – Оценка скорости шифрования

Рисунок 3.4 – Оценка скорости расшифрования

Также требовалось проанализировать время выполнения операций зашифрования/расшифрования при разных таблицах кодировки, и при увеличении числа членов ключевой последовательности.

Для проверки работоспособности было взято ФИО в формате Base64 «UnViYXNoZWsgQWxleGFuZHIgQWxleGFuZHJvdmljaA», также значение *z*=6. Результат выполнения можно увидеть на рисунке 3.5

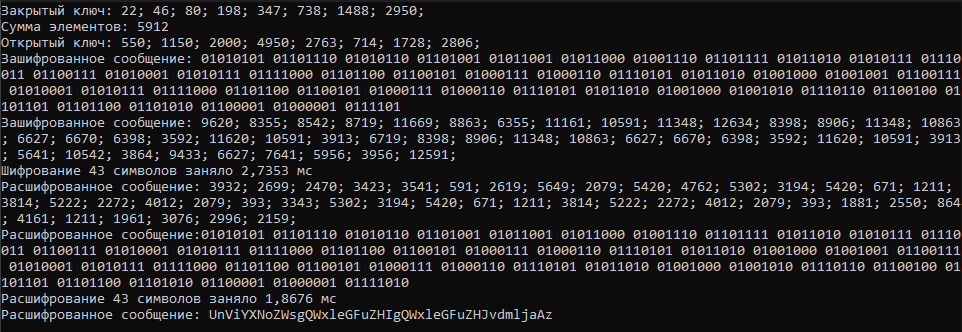


Рисунок 3.5 – Результат шифрования/дешифрования в Base64

Оценка скорости выполнения представлена на рисунках 3.6 – 3.7.

Рисунок 3.5 – Оценка скорости шифрования в Base64

Рисунок 3.5 – Оценка скорости дешифрования в Base64

**Вывод**

В ходе лабораторной работы были изучены и приобретены практические навыки разработки и использования приложений для реализации ассиметричных шифров.

Также было разработано авторское приложение в соответствии с целью лабораторной работы. Было проанализавана скорость выполнения шифрования и дешифрования в разны кодировках. По результатам они почти идентичны.

**Приложение 1**

Код функции GenerateSequence и ComputeSequence

public int[] GenerateSequence(int z)

{

int[] sequence = new int[z];

int sum = 0;

for (int i = 0; i < z; i++)

{

sequence[i] = randomizer.Next(sum, sum + 63);

sum += sequence[i];

}

return sequence;

}

public int[] ComputeSequence(int[] d, int a, int n, int z)

{

int[] e = new int[z];

for (int i = 0; i < z; i++)

{

e[i] = (d[i] \* a) % n;

}

return e;

}

**Приложение 2**

Код функции Encrypt и Decrypt

public int[] Encrypt(int[] d, string message, int z)

{

int[] result = new int[message.Length];

int j = 0;

Console.Write("Зашифрованное сообщение: ");

StringBuilder encryptedMessage = new StringBuilder();

StringBuilder binaryMessage = new StringBuilder();

foreach (char ch in message)

{

int total = 0;

string binary = '0' + Convert.ToString(ch, 2);

Console.Write($"{binary} ");

binaryMessage.Append(binary + " ");

for (int i = 0; i < binary.Length; i++)

{

if (binary[i] == '1')

{

total += d[i];

}

}

result[j] = total;

j++;

}

return result;

}

public string Decrypt(int[] d, int S, int z)

{

StringBuilder res = new StringBuilder();

StringBuilder res2 = new StringBuilder();

StringBuilder binaryMessage = new StringBuilder();

for (int i = z; i > 0; i--)

{

if (S >= d[i - 1])

{

res.Append('1');

S -= d[i - 1];

}

else

{

res.Append('0');

}

}

for (int i = res.Length - 1; i >= 0; i--)

{

res2.Append(res[i]);

binaryMessage.Append(res[i]);

}

Console.Write($"{res2} ");

return res2.ToString();

}