Учреждение образования

«Белорусский государственный технологический университет»

**Отчет по лабораторной работе №8**

по дисциплине: «Защита информации и надежность информационных систем»

Исполнитель:

Воликов Д. А., ФИТ 4-7

Руководитель:

Асс. Савельева М. Г.

Минск 2024

# Лабораторная работа №8. Исследование асимметричных шифров RSA и Эль-Гамаля

**Цель**: изучение и приобретение практических навыков разработки и использования приложений для реализации асимметричных шифров RSA и Эль-Гамаля.

**Задачи**:

1. Закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости асимметричных шифров RSA и Эль-Гамаля.
2. Разработать приложение для реализации асимметричного зашифрования/расшифрования на основе алгоритмов RSA и Эль-Гамаля.
3. Выполнить анализ криптостойкости асимметричных шифров RSA и Эль-Гамаля.
4. Оценить скорость зашифрования/расшифрования реализованных шифров.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

## Теоретические сведения

В силу этого практически все системы асимметричного зашифрования/расшифрования основаны либо на проблеме факторизации (среди них – RSA), либо на проблеме дискретного логарифмирования (среди них – Эль-Гамаля).

**Теорема 1**: основная теорема арифметики. Всякое натуральное число *N*, кроме 1, можно представить как произведение простых множителей:

*N* = *p*1*p*2*p*3...*p*z, *z* > 1.

**Определение 1**. Задача дискретного логарифмирования формулируется так: для данных целых чисел *а* и *b*, 1 < *а*, *b* < *n*, найти логарифм – такое целое число *х*, что

*a*x ≡ *b* (mod *n*),

если такое число существует.

По аналогии с вещественными числами используется обозначение  
*х* = loga*b*.

Теорема 2: китайская теорема об остатках. В общем случае если разложение числа *N* на простые множители представляет собой *p*1*p*2…*p*t (некоторые простые числа могут встречаться несколько раз), то система уравнений

*x* mod *p*i ≡ *a*i,

где *i* = 1, 2, …, *t*, имеет единственное решение: *x*, меньшее *N*.

Иными словами, число (меньшее, чем произведение нескольких простых чисел) однозначно определяется своими вычетами по модулю от этих простых чисел. Китайской теоремой об остатках можно воспользоваться для решения полной системы уравнений в том случае, если известно разложение числа *N* на простые множители.

Алгоритм RSA. Рассматриваемый алгоритм появился (1977 г.) после алгоритма ранца Меркла. Он стал первым полноценный алгоритмом с открытым ключом, который впоследствии стал одним из основных для шифрования и для электронных цифровых подписей.

Из всех предложенных алгоритмов с открытыми ключами RSA проще всего понять и реализовать. Он назван в честь трех его создателей: Рона Ривеста (Ron Rivest), Ади Шамира (Adi Shamir) и Леонарда Эдлемана (Leonard Adleman).

Как было отмечено, безопасность RSA основана на трудности разложения на множители больших чисел. Открытый и закрытый ключи являются функциями двух больших простых чисел. Предполагается, что восстановление открытого текста по шифртексту и открытому ключу эквивалентно разложению на множители двух больших чисел.

Для генерации двух ключей: тайного и открытого (а по сути ­– двух взаимосвязанных частей одного ключа, т. е. ключа, принадлежащего одному физическому лицу (или группе лиц), либо одному юридическому лицу), используются два больших случайных простых числа *p* и *q*. Для максимальной большей криптостойкости нужно выбирать *p* и *q* равной длины. Рассчитывается произведение: *n* = *p ⋅ q*. Это есть один из трех компонент ключа, состоящего из чисел *n*, *e*, *d*.

Затем случайным образом выбирается второй компонент ключа (открытый ключ или ключ зашифрования, *e*, такой что *e* и (*p* – 1) ⋅ (*q* – 1) являются взаимно простыми числами; вспомним, что (*p* – 1) ⋅ (*q* – 1) = *φ*(*n*) – функция Эйлера).

Наконец, расширенный алгоритм Евклида используется для вычисления третьего компонента ключа: ключа расшифрования *d* такого, что выполняется условие:

*e* ⋅ *d* ≡ 1 mod *φ*(*n*).

Таким образом, сформирован ключ, состоящий из трех чисел, которые в свою очередь образуют две вышеупомянутые взаимосвязанные части: открытый (публичный) ключ (*e*, *n*) и тайный ключ (*d*, *n*; на самом деле, как видим, тайным здесь является лишь первое из пары чисел).

Для зашифрования/расшифрования используется ключ получателя: отправитель шифрует сообщение открытым ключом, а получатель расшифровывает шифртекст своим тайным ключом.

Зашифрование. Если шифруется сообщение *М*, состоящее из *r* блоков: *m*1, *m*2, …, *m*i, …, *m*r, то шифртекст *С* будет состоять из такого же числа (*r*) блоков, представляемых числами:

*c*i ≡ (*m*i)e mod *n*.

Расшифрование. Для расшифрования каждого зашифрованного блока производится вычисление вида:

*m*i ≡ (*c*i)d mod *n*.

Алгоритм Эль-Гамаля. Предложен Т. Эль-Гамалем (T. El-Gamal) в 1985 г. Он может быть использован для решения трех основных криптографических задач: для зашифрования/расшифрования данных, для формирования цифровой подписи и для согласования общего ключа. Кроме того, возможны модификации алгоритма для схем проверки пароля, доказательства идентичности сообщения и другие варианты.

Рассматриваемый алгоритм отличается от алгоритма RSA несколькими параметрами и особенностями:

* генерацией ключевой информации и числом компонент, составляющих ключ;
* каждому блоку (символу) открытого сообщения в шифртексте на основе алгоритма Эль-Гамаля соответствуют 2 блока (в RSA – один-один);
* в алгоритме Эль-Гамаля при зашифровании используется число (обозначим его k), которое практически никак не связано с ключевой информацией получателя и которое принимает (по определению) различные значения при зашифровании различных блоков сообщения.

Генерация ключевой информации. Выбирается простое число *р*. Выбирается число (*g*, *g* < *p*), являющееся первообразным корнем числа *р* – очень важный элемент с точки зрения безопасности алгоритма.

Далее выбирается число *х* (*х* < p) и вычисляется последний компонент ключевой информации:

*y* ≡ *g*х mod *р*.

Владельцу сформированной ключевой информации, состоящей из 4 чисел, может посылаться некоторый шифртекст, созданный с использованием открытого ключа получателя: *p*, *g*, *y*. Расшифрование шифртекста получатель производит своим тайным ключом: *p*, *g*, *х*. Как видим, на самом деле тайным является лишь одно число (как и в RSA): *х*.

Зашифрование сообщения. Как ранее, предположим, что сообщение *М* = {*m*i}, где *m*i – *i*-й блок сообщения.

Зашифрование отправителем (каждого отдельного блока *m*i исходного сообщения) предусматривает использование, как это особо подчеркивалось выше, некоторого случайного числа *k* (1 < *k* < *p* – 1).

Блок шифртекста (*c*i) состоит из двух чисел – *а*i и *b*i:

*a*i ≡ *g*k mod *p*;

*b*i ≡ (*y*k ⋅ *m*i) mod *p*.

Здесь стал очевидным упомянутый недостаток алгоритма шифрования Эль-Гамаля: удвоение (реально – примерно в 1,5 раза) длины зашифрованного текста по сравнению с начальным текстом.

Случайное число *k* должно сразу после вычисления уничтожаться.

Расшифрование *c*i. Выполняется по следующей формуле:

*m*i ≡ (*b*i ⋅ (*a*i)p – x - 1) mod *p*,

где (*a*x)–1 – обратное значение числа *a*x по модулю *p*.

## Практическое задание

1. С помощью простого консольного приложения составить табличную или графическую форму зависимости времени вычисления параметра *у*, функционально заданного выражением вида:

*у* ≡ *a*x mod *n*,

от параметров: *а* (десятичные числа от 5 до 35; можно взять 1 или 2 числа), *х* (числа, желательно простые, из диапазона от 103 до 10100 ; для примера взять 5–10 чисел, равномерно распределенных в указанном диапазоне), *n* (для примера взять числа, в двоичном виде состоящие из 1024 и 2048 битов).

Для примера возьмём следующие числа (хотя в программе они всегда будут иметь случайное значение, но для примера возьмём определённые), которые представлены на рисунке ниже.

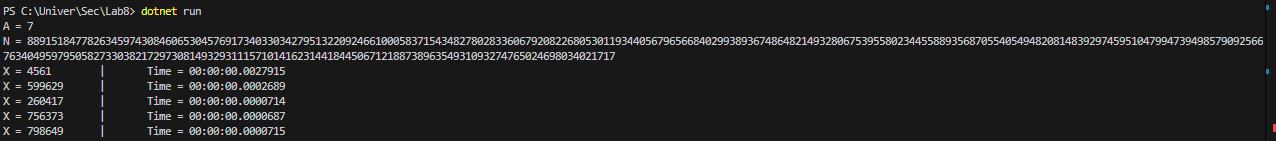


Рисунок 1 – Результат программы

Теперь оформим в виде графика зависимости времени от значения *x*.

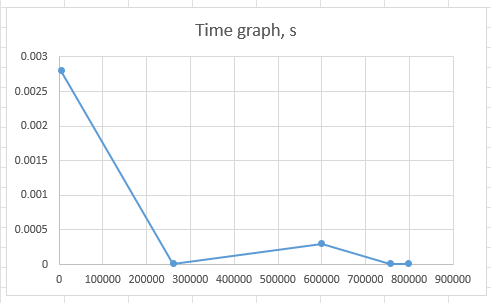


Рисунок 2 – График зависимости времени от значения *x*

Как можно заметить, время было максимальным только в начале, что можно объяснить поведением языка *C*# и платформы dotnet.

2. Разработать авторское оконное приложение в соответствии с целью лабораторной работы. При этом можно воспользоваться доступными библиотеками либо программными кодами.

В основе вычислений – кодировочные таблицы Base64 и ASCII.

Приложение должно реализовывать следующие операции:

• зашифрование и расшифрование текстовых документов на основе алгоритмов RSA и Эль-Гамаля;

• определение времени выполнения операций.

Исходный текст для зашифрования – собственные фамилия, имя, отчество. Для численного представления блоков текста можно в том числе пользоваться указанными выше кодировочными таблицами.

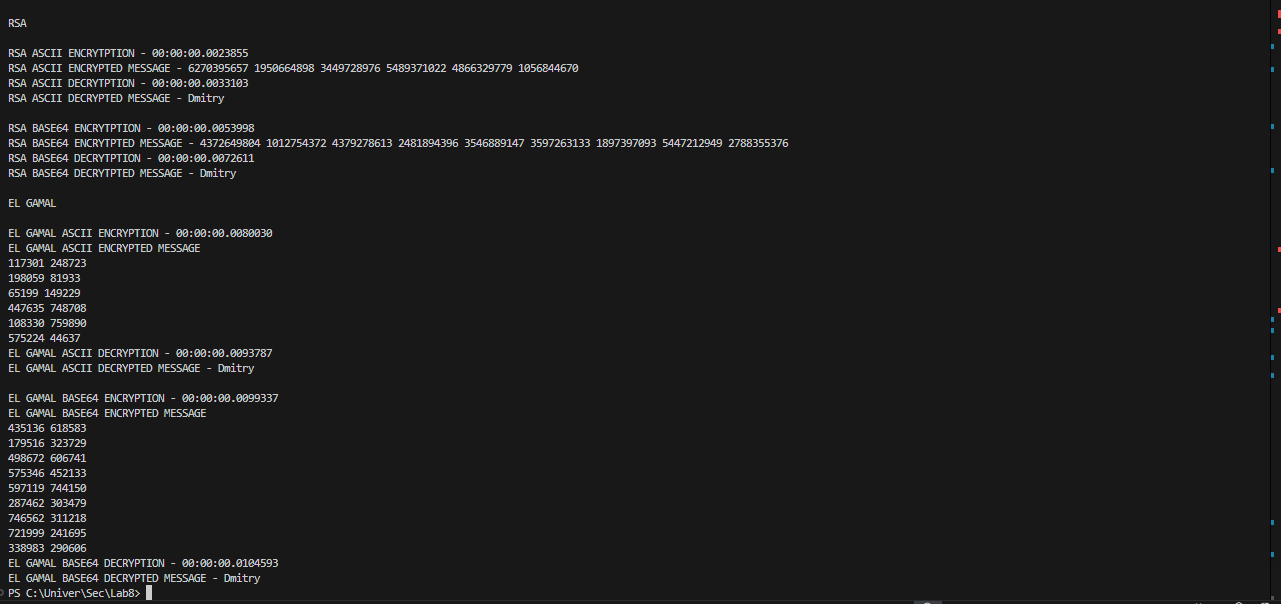


Рисунок 3 – Результат программы шифрования/расшифрования

На рисунке ниже представлен график зависимости скорости работы алгоритма RSA от использованной кодировки.

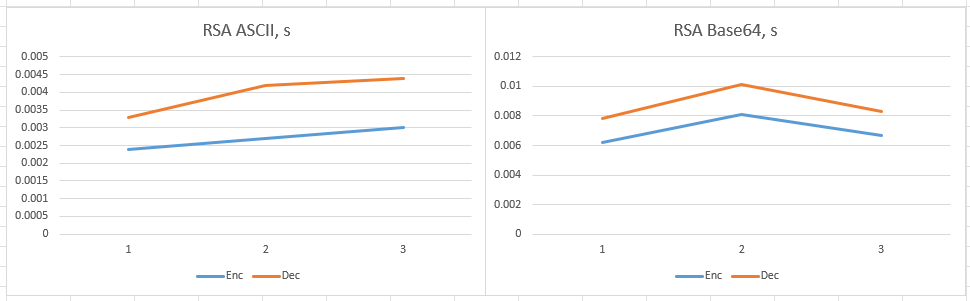


Рисунок 4 – График зависимости скорости работы алгоритма RSA

Также оценим зависимость скорости работы алгоритма El Gamal от использованной кодировки.

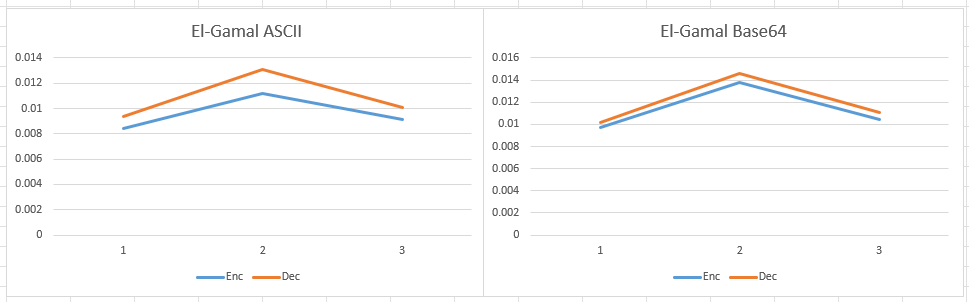


Рисунок 5 – График зависимости скорости работы алгоритма El-Gamal

3. Используя примерно одинаковый порядок ключевой информации, оценить производительность обоих алгоритмов и относительное изменение объемов криптотекстов (по отношению к объемам открытых текстов).

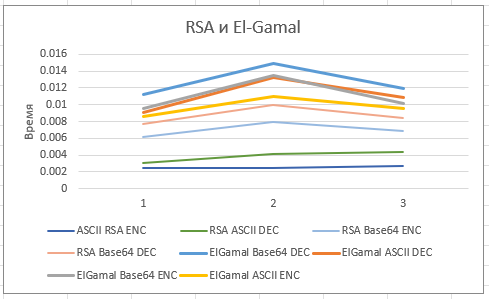


Рисунок 6 – Графики сравнения производительности алгоритмов

Сравнить объём шифртекста будет довольно тяжело, ибо кодировки ASCII и Base64 не имеют длины настолько больших значений, которые выдают алгоритмы. Поэтому оценка будет происходить по тому, сколько в памяти занимают типы данных.

В расчётах использовался целочисленный тип long – одно значение занимает 8 байт. Имея сообщение в 8 символов (т.е. 8 байт), в RSA получим отношение 1 к 8 битам в ASCII (или 1 к 11 битам), ибо для одного символа строки будет приходиться одно значение long. В ElGamal отношение будет 1 к 16 байтам в ASCII (или 1 к 22 битам), ибо зашифрованный символ сообщения представляется двумя значениями типа long.

Вывод: в ходе этой лабораторной работы я закрепил знания об алгоритмах асимметричного шифрования RSA и ElGamal, полученные на лекциях. На основе этого я разработал приложение, которое реализует зашифрование/расшифрование исходного сообщения в кодировках ASCII и Base64.

# Листинг «Лабораторная работа №8»

using System.Numerics;

using System.Text;

using Lab8.Encoders;

using Lab8.ExtensionMethods;

using Lab8.Generators.PrimeGenerator;

namespace Lab8.RSA

{

public class RSA

{

public long N { get; private set; } // module

public long E { get; private set; } // public key

private readonly long D = 0; // private key

public RSA()

{

N = (long)PrimeGenerator.GetRandomPrime() \* PrimeGenerator.GetRandomPrime();

E = N.EilersFunction().GetCoprimeBelow();

D = E.GetReversed(N.EilersFunction());

}

public long[] AsciiEncrypt(string message)

{

return Encoding.ASCII.GetBytes(message)

.Select(b => (long)BigInteger.ModPow(b, E, N))

.ToArray();

}

public string AsciiDecrypt(long[] message)

{

return Encoding.ASCII.GetString(

message.Select(m => (byte)BigInteger.ModPow(m, D, N))

.ToArray()

);

}

public long[] Base64Encrypt(string message)

{

return Encoding.UTF8.GetBytes(Base64.Encode(message))

.Select(b => (long)BigInteger.ModPow(b, E, N))

.ToArray();

}

public string Base64Decrypt(long[] message)

{

return Base64.Decode(

Encoding.UTF8.GetString(

message.Select(m => (byte)BigInteger.ModPow(m, D, N)).ToArray()

)

);

}

}

}

Листинг 1 – Код алгоритма RSA

using System.Numerics;

using System.Text;

using Lab8.Encoders;

using Lab8.ExtensionMethods;

using Lab8.Generators.PrimeGenerator;

namespace Lab8.ElGamal

{

public class ElGamal

{

public long P { get; private set; } // part of public key (random prime number)

public long Q { get; private set; } // part of public key (primitive root of P)

public long Y { get; private set; } // part of public key

private readonly long X; // private key

public ElGamal()

{

P = PrimeGenerator.GetRandomPrime();

Q = P.GetPrimitiveRoot();

X = new Random().NextInt64(2, P - 1);

Y = (long)BigInteger.ModPow(Q, X, P);

}

public ElGamalEncryptedByte[] AsciiEncrypt(string message)

{

return Encoding.ASCII.GetBytes(message)

.Select(b => {

long K = PrimeGenerator.GetRandomPrime();

return new ElGamalEncryptedByte

{

A = (long)BigInteger.ModPow(Q, K, P),

B = (long)(BigInteger.ModPow(Y, K, P) \* b % P)

};

})

.ToArray();

}

public string AsciiDecrypt(ElGamalEncryptedByte[] message)

{

return Encoding.ASCII.GetString(

message

.Select(b => (byte)(BigInteger.ModPow(b.A, P - 1 - X, P) \* b.B % P))

.ToArray()

);

}

public ElGamalEncryptedByte[] Base64Encrypt(string message)

{

return Encoding.UTF8.GetBytes(Base64.Encode(message))

.Select(b => {

long K = PrimeGenerator.GetRandomPrime();

return new ElGamalEncryptedByte

{

A = (long)BigInteger.ModPow(Q, K, P),

B = (long)(BigInteger.ModPow(Y, K, P) \* b % P)

};

})

.ToArray();

}

public string Base64Decrypt(ElGamalEncryptedByte[] message)

{

return Base64.Decode(

Encoding.UTF8.GetString(

message

.Select(b => (byte)(BigInteger.ModPow(b.A, P - 1 - X, P) \* b.B % P))

.ToArray()

)

);

}

public record ElGamalEncryptedByte

{

public long A { get; set; } // A parameter

public long B { get; set; } // B parameter

}

}

}

Листинг 2 – Код алгоритма ElGamal

using System.Numerics;

using System.Security.Cryptography;

using Lab8.ElGamal;

using Lab8.Generators.PrimeGenerator;

BigInteger a = new(7);

byte[] bytes = new byte[128];

using var rng = RandomNumberGenerator.Create();

rng.GetBytes(bytes);

BigInteger n = new(bytes);

if (n.Sign == -1) n = BigInteger.Multiply(n, -1);

BigInteger[] xArray = [

PrimeGenerator.GetRandomPrime(),

PrimeGenerator.GetRandomPrime(),

PrimeGenerator.GetRandomPrime(),

PrimeGenerator.GetRandomPrime(),

PrimeGenerator.GetRandomPrime(),

];

Console.WriteLine($"A = {a}\nN = {n}");

foreach (var item in xArray)

{

var timer = System.Diagnostics.Stopwatch.StartNew();

var y = BigInteger.ModPow(a, item, n);

timer.Stop();

Console.WriteLine($"X = {item}\t|\tTime = {timer}");

}

var message = "Dmitry";

var rsaEncryptor = new Lab8.RSA.RSA();

var elGamalEncryptor = new ElGamal();

Console.WriteLine("\nRSA\n");

var watch = System.Diagnostics.Stopwatch.StartNew();

var rsaAsciiEncryptedMessage = rsaEncryptor.AsciiEncrypt(message);

watch.Stop();

Console.WriteLine($"RSA ASCII ENCRYTPTION - {watch}");

Console.WriteLine($"RSA ASCII ENCRYPTED MESSAGE - {string.Join(" ", rsaAsciiEncryptedMessage.Select(c => c.ToString()))}");

watch.Start();

var rsaAsciiDecryptedMessage = rsaEncryptor.AsciiDecrypt(rsaAsciiEncryptedMessage);

watch.Stop();

Console.WriteLine($"RSA ASCII DECRYTPTION - {watch}");

Console.WriteLine($"RSA ASCII DECRYPTED MESSAGE - {rsaAsciiDecryptedMessage}\n");

watch.Start();

var rsaBase64EncryptedMessage = rsaEncryptor.Base64Encrypt(message);

watch.Stop();

Console.WriteLine($"RSA BASE64 ENCRYTPTION - {watch}");

Console.WriteLine($"RSA BASE64 ENCRYTPTED MESSAGE - {string.Join(" ", rsaBase64EncryptedMessage.Select(c => c.ToString()))}");

watch.Start();

var rsaBase64DecryptedMessage = rsaEncryptor.Base64Decrypt(rsaBase64EncryptedMessage);

watch.Stop();

Console.WriteLine($"RSA BASE64 DECRYTPTION - {watch}");

Console.WriteLine($"RSA BASE64 DECRYTPTED MESSAGE - {rsaBase64DecryptedMessage}");

Console.WriteLine("\nEL GAMAL\n");

watch.Start();

var elGamalAsciiEncryptedMessage = elGamalEncryptor.AsciiEncrypt(message);

watch.Stop();

Console.WriteLine($"EL GAMAL ASCII ENCRYPTION - {watch}");

Console.WriteLine($"EL GAMAL ASCII ENCRYPTED MESSAGE\n{string.Join('\n', elGamalAsciiEncryptedMessage.Select(b => b.A + " " + b.B).ToArray())}");

watch.Start();

var elGamalAsciiDecryptedMessage = elGamalEncryptor.AsciiDecrypt(elGamalAsciiEncryptedMessage);

watch.Stop();

Console.WriteLine($"EL GAMAL ASCII DECRYPTION - {watch}");

Console.WriteLine($"EL GAMAL ASCII DECRYPTED MESSAGE - {elGamalAsciiDecryptedMessage}\n");

watch.Start();

var elGamalBase64EncryptedMessage = elGamalEncryptor.Base64Encrypt(message);

watch.Stop();

Console.WriteLine($"EL GAMAL BASE64 ENCRYPTION - {watch}");

Console.WriteLine($"EL GAMAL BASE64 ENCRYPTED MESSAGE\n{string.Join('\n', elGamalBase64EncryptedMessage.Select(b => b.A + " " + b.B).ToArray())}");

watch.Start();

var elGamalBase64DecryptedMessage = elGamalEncryptor.Base64Decrypt(elGamalBase64EncryptedMessage);

watch.Stop();

Console.WriteLine($"EL GAMAL BASE64 DECRYPTION - {watch}");

Console.WriteLine($"EL GAMAL BASE64 DECRYPTED MESSAGE - {elGamalBase64DecryptedMessage}");

Рисунок 3 – Точка входа в программу