Министерство образования Республики Беларусь

Учреждение образования

Белорусский государственный университет информатики и

радиоэлектроники

Кафедра ЭВМ

Отчет

по лабораторной работе № 4

Криптографическая подпись Фиат-Шамира

Выполнили: Проверил:

студенты группы 050503 доцент, к.т.н

Торчилов П.Р. Кобяк И. П.

Белько С.Д.

Минск 2014

1. **Исходные данные.**

Сущность метода заключается в следующем:

Пусть h – хэш-функция, или функция формирования сигнатур, преобразующая исходное сообщение µ в строку символов длиной l.

1. Для формирования подписи выберем 2 простых числа ξ1 и ξ1, и определим модуль для сравнения составляющих подписи m = ξ1 · ξ1;
2. Выбираем случайное число 0 < α ≤ m – 1;
3. Вычисляем β ≡ α2 (mod m);
4. Вычисляем хэш-функцию для сообщения µ и β, которые представляют собой набор символов в двоичной системе счисления.

Полученное значение S используется как первый элемент подписи;

1. В качестве секретного ключа абонент источник генерирует ряд различный случайных чисел ai, взаимно простых к m;
2. Рассчитываем второй элемент подписи t:

t = α · (mod m);

1. Открытый ключ B определяется путем решения сравнения при заданных Ai:

bi ≡ () 2  (mod m).

Алгоритм проверки подписи состоит в выполнении следующих действий:

1. По открытому ключу B по модулю m и значению t определяется параметр ω:

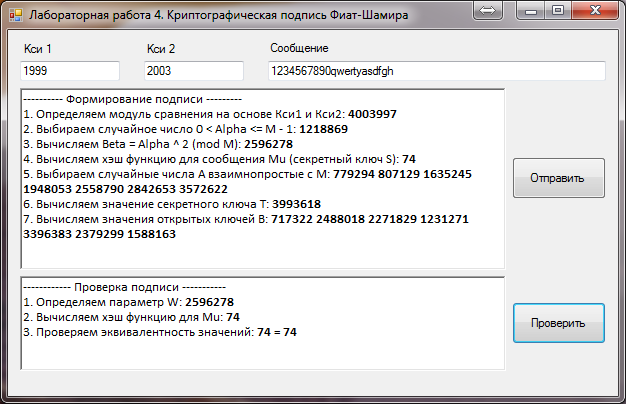
ω = t2 · (mod m);

1. По известному сообщению строится хэш-функция h(µ, ω);
2. Сравнивают S и h(µ, ω) и в случае совпадения считают, что данные переданы верно.
3. **Задание.**

Выбрать необходимые параметры криптографической системы ξ1 и ξ1. Сформировать сообщение. На основе выбранных параметров рассчитать необходимые элементы подписи и передать их вместе с сообщением.

Абонент-приемник на основе полученной подписи и сообщения должен сделать вывод о том, верно ли переданы данные.

1. **Результаты работы программы.**

****

1. **Листинг программы.**

public class FiatShamir

{

public long Ksi1 { get; private set; }

public long Ksi2 { get; private set; }

public long M { get; private set; }

public long Alpha { get; private set; }

public long Beta { get; private set; }

public long S { get; private set; }

public long[] A { get; private set; }

public long T { get; private set; }

public long[] B { get; private set; }

public long W { get; private set; }

public long Hash { get; private set; }

private readonly Random random;

public FiatShamir(long ksi1, long ksi2)

{

Ksi1 = ksi1;

Ksi2 = ksi2;

random = new Random();

}

public long[] GenerateSignature(String mu)

{

//1. Определяем модуль сравнения на основе Кси1 и Кси2

M = Ksi1 \* Ksi2;

//2. Выбираем случайное число 0 < Alpha <= M - 1

Alpha = random.Next(1, (int) M);

//3. Вычисляем Beta = Alpha ^ 2 (mod M)

Beta = (long) Math.Pow(Alpha, 2) % M;

//4. Вычисляем хэш функцию для сообщения Mu (секретный ключ S)

S = GetHashFunction(mu, Beta);

var hashBinary = Convert.ToString(S, 2);

//5. Выбираем случайные числа A взаимнопростые с M

A = GetANumbers(hashBinary.Length);

//6. Вычисляем занчение секретного ключа T

T = GetTKey(Alpha, A, hashBinary);

//7. Вычисляем значения открытых ключей B

B = GetBKeys(A);

return new[] { S, T };

}

public bool ValidateSignature(long[] keys, String mu)

{

//1. Определяем параметр w

W = GetWParam(keys[0], keys[1], B);

//2. Вычисляем хэш функцию для Mu

Hash = GetHashFunction(mu, W);

//3. Проверяем эквивалентность значений

return Hash == keys[0];

}

private long GetWParam(long s, long t, long[] b)

{

var sBinary = Convert.ToString(s, 2);

BigInteger w = t \* t;

for (var i = 0; i < sBinary.Length; i++)

{

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

{

w \*= b[i];

}

}

return (long) (w % M);

}

private long[] GetBKeys(long[] a)

{

var b = new long[a.Length];

for (var i = 0; i < a.Length; i++)

{

var value = a[i] \* a[i];

b[i] = GetB(value);

}

return b;

}

private long GetB(long divider)

{

var quotients = new List<long>();

var dividend = M;

while (divider != 0)

{

var quotient = dividend / divider;

quotients.Add(quotient);

var temp = dividend;

dividend = divider;

divider = temp - (quotient \* divider);

}

var ps = new List<long>();

var ps0 = 1;

ps.Add(ps0);

var ps1 = quotients[0] \* ps[0];

ps.Add(ps1);

for (var i = 1; i < quotients.Count; i++)

{

var pi = ps[i] \* quotients[i] + ps[i - 1];

ps.Add(pi);

}

var x = Convert.ToInt64(Math.Pow(-1, quotients.Count - 1));

x \*= ps[ps.Count - 2];

if (x < 0)

{

x += M;

}

return x;

}

private long GetTKey(long alpha, long[] a, String hashBinary)

{

BigInteger t = alpha;

for (var i = 0; i < hashBinary.Length; i++)

{

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

{

t \*= a[i];

}

}

return (long) (t % M);

}

private long[] GetANumbers(long length)

{

var result = new long[length];

var candidates = new List<int>();

for (var i = 2; i < M; i++)

{

if (GetGreatestCommonDivisor(i, M) == 1)

{

candidates.Add(i);

}

}

if (candidates.Count < length)

{

throw new ArgumentException("Недостаточно взаимно простых чисел.");

}

for (var i = 0; i < length; i++)

{

var index = random.Next(0, candidates.Count);

result[i] = candidates[index];

candidates.RemoveAt(index);

}

return result.OrderBy(x => x).ToArray();

}

private long GetHashFunction(String mu, long beta)

{

var coder = new Coder();

var formattedMsg = coder.FormatMessage(mu);

var codedMsg = coder.CodeMessage(formattedMsg);

var betaBinary = Convert.ToString(beta, 2);

while (betaBinary.Length < Coder.CodeLength)

{

betaBinary = "0" + betaBinary;

}

return GetHash(codedMsg + betaBinary);

}

private long GetHash(String mu)

{

var hashResult = mu.Count(character => character == '1');

return hashResult;

}

private long GetGreatestCommonDivisor(long a, long b)

{

return b == 0 ? a : GetGreatestCommonDivisor(b, a % b) ;

}

}

private void ButtonSendClick(object sender, EventArgs e)

{

var ksi1String = textBoxKsi1.Text;

var ksi2String = textBoxKsi2.Text;

if (ValidateKsiValue(ksi1String, 1) && ValidateKsiValue(ksi2String, 2))

{

var ksi1 = Convert.ToInt64(textBoxKsi1.Text);

var ksi2 = Convert.ToInt64(textBoxKsi2.Text);

message = textBoxMessage.Text;

fiatShamir = new FiatShamir(ksi1, ksi2);

keys = fiatShamir.GenerateSignature(message);

ShowParams();

}

}

private void ButtonVerifyClick(object sender, EventArgs e)

{

if (keys != null && keys.Any())

{

if (fiatShamir != null)

{

var isValid = fiatShamir.ValidateSignature(keys, message);

ShowValidateParams(isValid);

}

}

}

1. **Вывод.**

В результате выполнения лабораторной работы была написана программа, которая рассчитывает криптографическую подпись для передаваемого сообщения по алгоритму Фиат-Шамира.

Данная подпись позволяет однозначно идентифицировать отправителя, тем самым реализую аутентификацию, а также обнаруживать ошибки или искажения, возникшие при передачи сообщения в канале связи.