

**Министерство науки и высшего образования
Липецкий государственный технический университет**

Факультет автоматизации и информатики
Кафедра прикладной математики

Отчет по лабораторной работе № 1
по дисциплине «Безопасности компьютерных систем»
на тему «Симметричные криптосистемы: поточные шифры»

Студент

Группа ПМ-19-1

Руководитель

к.т.н., доцент

учёная степень, учёное звание

подпись, дата

подпись, дата

Пестова А.Ю.

фамилия, инициалы

Сысоев А.С.

фамилия, инициалы

Липецк 2023 г.

Оглавление

Задание кафедры	2
1 Теоретическая часть	3
1.1 Линейный конгруэнтный генератор	3
1.2 Генератор Blum Blum Shub	3
1.3 Синхронное поточное шифрование	4
1.4 Асинхронное поточное шифрование	4
2 Практическая часть	5
2.1 Исходные данные	5
2.2 Результаты запуска программы	5
Заключение	7
Приложения	8
Приложения	9
Приложения	10
Приложения	13

Задание кафедры

Реализовать алгоритмы синхронного поточного и асинхронного поточного шифрования. В качестве генератора гаммы использовать линейный конгруэнтный генератор, а так же один из предложенных по выбору: генератор квадратичных вычетов, генератор BBS, генератор М-последовательностей, генератор последовательностей Голда. На контрольном примере проверить правильность работы алгоритмов шифрования и дешифрования.

1 Теоретическая часть

1.1 Линейный конгруэнтный генератор

Линейный конгруэнтный генератор псевдослучайных последовательностей вырабатывает последовательности псевдослучайных чисел, описываемые соотношением:

$$\gamma_{i+1} = (a \cdot \gamma_i + b) \mod e,$$

где a, b, e — некоторые константы, γ_0 — некоторая величина, выбранная в качестве порождающего ключа. Тройка (γ_0, a, b) порождает ключ.

Датчик генерирует псевдослучайные числа с определенным периодом повтора, равным e , не зависящим от чисел a и b .

При реализации следует выбирать $m = 2^k + 1$ или $m = 2k$, k — длина машинного слова в битах (k равно 16 или 32).

Статистические свойства получаемой последовательности полностью определяются выбором констант a и b при заданной разрядности k .

1.2 Генератор Blum Blum Shub

Простой, но эффективный метод создания генератора псевдослучайных чисел назван Blum Blum Shub (BBS) по имени его трех изобретателей.

BBS использует уравнение квадратичного вычета, но это - псевдослучайный генератор бит вместо генератора псевдослучайных чисел; он генерирует последовательность битов (0 или 1).

Ниже приведены шаги генерации:

1. Найти два больших простых числа p, q в форме $4k + 3$, где k — целое число (p, q — являются конгруэнтными $3 \mod 4$).
2. Выбрать модуль $n = p \cdot q$.
3. Выбрать случайное целое число r , которое является взаимно-простым с n .
4. Вычислить начальное число как $x_0 = r^2 \mod n$.
5. Генерировать последовательность $x_{i+1} = x_i^2 \mod n$.
6. Взять самый младший бит сгенерированного случайного целого числа как случайный бит.

1.3 Синхронное поточное шифрование

Гамма получается вне зависимости от исходного и шифрованного текстов.

Шифр вырабатывает гамму на основе секретного ключа. Она складывается с открытым текстом, и результат посылается другому абоненту и расшифровывается аналогично.

Блок, вырабатывающий гамму, называется генератором гаммы (псевдослучайным генератором).

Для i -го шага алгоритм имеет вид:

$$tmp_i = inc_i \oplus \gamma_i,$$

где inc_i — символ открытого текста, tmp_i — символ закрытого текста, γ_i — генерирование числа конгруэнтным или BBS генератором.

В результате выполнения алгоритма получаем закрытый текст. Повторное применение приведённого алгоритма преобразует закрытый текст обратно в открытый.

1.4 Асинхронное поточное шифрование

Каждый знак ключевого потока определяется фиксированным числом предшествующих знаков шифртекста.

Для $i < t$ -го шага алгоритм имеет вид:

$$tmp_i = inc_i \oplus \gamma_i,$$

где inc_i — символ открытого текста, tmp_i — символ закрытого текста, γ_i — генерирование числа конгруэнтным или BBS генератором.

Для $i \geq t$ -го шага в случае шифровки алгоритм имеет вид:

$$\gamma'_i = tmp_0 \oplus \dots \oplus tmp_i \oplus \gamma_i; tmp_i = inc_i \oplus \gamma'_i,$$

где inc_i — символ открытого текста, tmp_i — символ закрытого текста, γ_i — генерирование числа конгруэнтным или BBS генератором.

Для $i \geq t$ -го шага в случае дешифровки алгоритм имеет вид:

$$\gamma'_i = inc_0 \oplus \dots \oplus inc_i \oplus \gamma_i; tmp_i = inc_i \oplus \gamma'_i,$$

где inc_i — символ открытого текста, tmp_i — символ закрытого текста, γ_i — генерирование числа конгруэнтным или BBS генератором.

2 Практическая часть

2.1 Исходные данные

В таблице 1 приведены исходные данные, с которыми запускалась программа:

Таблица 1 – Исходные данные

Переменная	Значение
a	3
b	5
e	32
сообщение	Hello, World! It is GoLang!
γ_0	z
t	2
p	11
q	23
r	1024

2.2 Результаты запуска программы

```
Синхронный поточный шифр
Длина расшифрованного сообщения: 27
Длина зашифрованного сообщения: 27
Расшифрованное сообщение (в байтах): [72 101 108 108 111 44 32 87 111 114 108 100 33 32 73 116 32 105 115 32 71 111 76 97 110 103 33]
Зашифрованное сообщение (в байтах): [50 118 114 115 109 39 38 64 101 113 98 107 51 59 95 115 58 122 109 63 69 100 74 118 100 100 47]
Расшифрованное сообщение: `Hello, World! It is GoLang!`
Зашифрованное сообщение: `2vrsm'&@eqbk3;_s:zm?EdJvdd/`
```

Рисунок 1 – Синхронный поточный шифр с линейным конгруэнтным генератором

```

Синхронный поточный шифр

Длина расшифрованного сообщения: 27
Длина зашифрованного сообщения: 27
Расшифрованное сообщение (в байтах): [72 101 108 108 111 44 32 87 111 114 108 100 33 32 73 116 32 105 115 32 71 111 76 97 110 103 33]
Зашифрованное сообщение (в байтах): [50 118 114 115 109 39 38 64 101 113 98 107 51 59 95 115 58 122 109 63 69 100 74 118 100 100 47]
Расшифрованное сообщение `Hello, World! It is GoLang!`
Зашифрованное сообщение: `2vrsm'&@eqbk3;_s:zm?EdJvdd/`

```

Рисунок 2 – Синхронный поточный шифр с генератором BBS

```

Асинхронный поточный шифр

Длина расшифрованного сообщения: 27
Длина зашифрованного сообщения: 27
Расшифрованное сообщение (в байтах): [72 101 108 108 111 44 32 87 111 114 108 100 33 32 73 116 32 105 115 32 71 111 76 97 110 103 33]
Зашифрованное сообщение (в байтах): [50 118 54 68 113 117 75 69 65 33 99 84 122 106 120 70 87 12 8 77 113 115 9 116 95 45 61]
Расшифрованное сообщение `Hello, World! It is GoLang!`
Зашифрованное сообщение: `2v6DquKEA!cTzjxFW
                                Mqs   t_--`

```

Рисунок 3 – Асинхронный поточный шифр с линейным конгруэнтным генератором

```

Асинхронный поточный шифр, BBS

Длина расшифрованного сообщения: 27
Длина зашифрованного сообщения: 27
Расшифрованное сообщение (в байтах): [72 101 108 108 111 44 32 87 111 114 108 100 33 32 73 116 32 105 115 32 71 111 76 97 110 103 33]
Зашифрованное сообщение (в байтах): [216 150 70 78 11 5 65 63 49 43 25 36 112 16 8 76 45 124 2 55 1 121 115 4 85 87 77]
Расшифрованное сообщение `Hello, World! It is GoLang!`
Зашифрованное сообщение: ``FN
                                A?1+$L-!7ysUWM`

```

Рисунок 4 – Асинхронный поточный шифр с генератором BBS

Заключение

В ходе выполнения данной работы были реализованы алгоритмы синхронного и асинхронного поточного шифрования. В качестве генератора были использованы линейный конгруэнтный генератор, а так же генератор BBS. На контрольном примере были проверены правильность и корректность работы приведённых алгоритмов шифрования и дешифрования.

Приложение А

(справочное)

Реализация алгоритма линейного конгруэнтного генератора

```
package stream

type LinearCongruent struct {
    a      int
    b      int
    e      int
    gamma  byte
}

func (l *LinearCongruent) Gamma() byte {
    return l.gamma
}

func (l *LinearCongruent) NextGamma() {
    l.gamma = byte((l.a*int(l.gamma) + l.b) % l.e)
}

func (l *LinearCongruent) SetGamma(gamma byte) {
    l.gamma = gamma
}

type LinearCongruentGenerator interface {
    Gamma() byte
    NextGamma()
    SetGamma(gamma byte)
}

func NewLinearCongruentGenerator(a, b, e int, gamma string) LinearCongruentGenerator {
    return &LinearCongruent{
        a:      a,
        b:      b,
        e:      e,
        gamma:  []byte(gamma)[0],
    }
}
```

Приложение Б

(справочное)

Реализация алгоритма генератора Blum Blum Shub

```
package stream

type BBS struct {
    gamma byte
    n      int
}

type BBSInterface interface {
    Gamma() byte
    NextGamma()
    SetGamma(gamma byte)
}

func NewBBSInterface(r, p, q int) BBSInterface {
    return &BBS{
        gamma: byte(r * r % (p * q)),
        n:     p * q,
    }
}

func (B *BBS) Gamma() byte {
    return B.gamma
}

func (B *BBS) NextGamma() {
    B.gamma = byte(int(B.gamma) * int(B.gamma) % B.n)
}

func (B *BBS) SetGamma(gamma byte) {
    B.gamma = gamma
}
```

Приложение В

(справочное)

Реализация алгоритмов синхронного и асинхронного поточного шифрования

```
package stream

type Encrypt struct {
    a      int
    b      int
    e      int
    t      int
    p      int
    q      int
    r      int
    gamma  string
}

type EncryptInterface interface {
    SyncEncrypt(message string, bbs bool) []byte
    AsyncEncrypt(message string, decode, bbs bool) []byte
}

func NewEncryptInterface(a, b, e, t, p, q, r int, gamma string) EncryptInterface {
    return &Encrypt{
        a:      a,
        b:      b,
        e:      e,
        t:      t,
        p:      p,
        q:      q,
        r:      r,
        gamma:  gamma,
    }
}

func (e *Encrypt) SyncEncrypt(message string, bbs bool) []byte {
    var (
        inc      = make([]byte, 0)
        tmp      = make([]byte, 0)
        str      = []byte(message)
        lCongruent LinearCongruentGenerator
        bbsInterface BBSInterface
    )

    if !bbs {
        lCongruent = NewLinearCongruentGenerator(e.a, e.b, e.e, e.gamma)
    } else {
        bbsInterface = NewBBSInterface(e.r, e.p, e.q)
    }

    for i, val := range str {
        inc = append(inc, val)
    }
}
```

```

        if !bbs {
            tmp = append(tmp, inc[i]^lCongruent.Gamma())
            lCongruent.NextGamma()
            continue
        }
        tmp = append(tmp, inc[i]^bbsInterface.Gamma())
        bbsInterface.NextGamma()
    }
    return tmp
}

func (e *Encrypt) AsyncEncrypt(message string, decode, bbs bool) []byte {
    var (
        inc          = make([]byte, 0)
        tmp          = make([]byte, 0)
        str          = []byte(message)
        lCongruent   LinearCongruentGenerator
        bbsInterface BBSInterface
        _gamma       byte
    )

    if !bbs {
        lCongruent = NewLinearCongruentGenerator(e.a, e.b, e.e, e.gamma)
    } else {
        bbsInterface = NewBBSInterface(e.r, e.p, e.q)
    }

    for i, val := range str {
        inc = append(inc, val)
        if i < e.t {
            if !bbs {
                tmp = append(tmp, inc[i]^lCongruent.Gamma())
                lCongruent.NextGamma()
                continue
            }
            tmp = append(tmp, inc[i]^bbsInterface.Gamma())
            bbsInterface.NextGamma()
            continue
        }

        if !decode {
            _gamma = tmp[0]
        } else {
            _gamma = inc[0]
        }

        for j := 1; j < i; j++ {
            if !decode {
                _gamma = tmp[j] ^ _gamma
                continue
            }
            _gamma = inc[j] ^ _gamma
        }

        if !bbs {
            _gamma = lCongruent.Gamma() ^ _gamma
            lCongruent.SetGamma(_gamma)
            tmp = append(tmp, inc[i]^lCongruent.Gamma())
        } else {
            _gamma = bbsInterface.Gamma() ^ _gamma
            bbsInterface.SetGamma(_gamma)
        }
    }
}

```

```
        tmp = append(tmp, inc[i]^bbsInterface.Gamma())
    }
}
return tmp
}
```

Приложение Г

(справочное)

Реализация основного запускаемого файла

```
package main

import (
    "fmt"
    "github.com/evgen1067/computer_security/stream"
)

func main() {
    var (
        a      = 3
        b      = 5
        e      = 32
        message = "Hello, World! It is GoLang!"
        gamma  = "z"
        t      = 2
        p      = 11
        q      = 23
        r      = 1024
    )

    fmt.Println(" ")

    enc := stream.NewEncryptInterface(a, b, e, t, p, q, r, gamma)

    code := enc.SyncEncrypt(message, false)
    decoding := enc.SyncEncrypt(string(code), false)
    printResults(decoding, code)

    fmt.Println("\n\tn\t\t\t\t\t, BBS")

    code = enc.SyncEncrypt(message, true)
    decoding = enc.SyncEncrypt(string(code), true)
    printResults(decoding, code)

    fmt.Println("\n\tn\t\t\t\t\t")

    code = enc.AsyncEncrypt(message, false, false)
    decoding = enc.AsyncEncrypt(string(code), true, false)
    printResults(decoding, code)

    fmt.Println("\n\tn\t\t\t\t\t, BBS")

    code = enc.AsyncEncrypt(message, false, true)
    decoding = enc.AsyncEncrypt(string(code), true, true)
    printResults(decoding, code)
}

func printResults(decoding, code []byte) {
    fmt.Printf(
        "\tn\t\t\t\t\t: %v\tn\t\t\t\t\t: %v",
        len(decoding),
    )
}
```

}