ЛАБОРАТОРНАЯ РАБОТА №5

**Тема:** Изучение симметричных алгоритмов шифрования

1. **Текст программы, реализующей режим простой вставки и гаммирования**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 1 | package main | | 2 |  | | 3 | import ( | | 4 | "encoding/base64" | | 5 | "fmt" | | 6 | "math/rand" | | 7 | "bytes" | | 8 | "math" | | 9 | "encoding/binary" | | 10 | "bufio" | | 11 | "strings" | | 12 | "os" | | 13 | ) | | 14 |  | | 15 | *type* Gost *struct* { | | 16 | table [8][16]*byte* | | 17 | } | | 18 |  | | 19 | *type* Gost\_Mode *uint* | | 20 |  | | 21 | *const* ( | | 22 | GOST\_REPLACEMENT = Gost\_Mode(1) | | 23 | GOST\_XORING      = Gost\_Mode(2) | | 24 | ) | | 25 |  | | 26 | *func* (*g* Gost) encryptStr(*data*, *key* *string*, *mode* Gost\_Mode) *string* { | | 27 | return base64.StdEncoding.EncodeToString(g.encrypt([]*byte*(data), []*byte*(key), mode)) | | 28 | } | | 29 |  | | 30 | *func* (*g* Gost) decryptStr(*data*, *key* *string*, *mode* Gost\_Mode) *string* { | | 31 | base64decoded, \_ := base64.StdEncoding.DecodeString(data) | | 32 | return *string*(g.decrypt([]*byte*(base64decoded), []*byte*(key), mode)) | | 33 | } | | 34 |  | | 35 | *func* (*g* Gost) encrypt(*data*, *key* []*byte*, *mode* Gost\_Mode) []*byte* { | | 36 | closedData := new(bytes.Buffer) | | 37 | switch mode { | | 38 | case GOST\_REPLACEMENT: | | 39 | for len(data)%8 > 0 { | | 40 | data = append(data, 0) | | 41 | } | | 42 | for i := 0; i < len(data); i+=8 { | | 43 | blockBytes := data[i : i+8] | | 44 | block := g.encryptionCycle(g.bytesToUint64(blockBytes), key) | | 45 | closedData.Write(g.uint64ToBytes(block)) | | 46 | } | | 47 | default: | | 48 | fallthrough | | 49 | case GOST\_XORING: | | 50 | rand.Seed(179426549) | | 51 | openBlock := *uint64*(0) | | 52 | for i := 0; i < len(data); i++ { | | 53 | openBlock |= *uint64*(data[i]) << *uint*(8\*(i%8)) | | 54 | if ((i+1)%8 == 0 && i != 0) || i == len(data)-1 { | | 55 | buffer := *uint64*(rand.Int31()) | | 56 | gamma := g.encryptionCycle(buffer, key) | | 57 | closedData.Write(g.uint64ToBytes(openBlock^gamma)) | | 58 | openBlock = 0 | | 59 | } | | 60 | } | | 61 | } | | 62 | return closedData.Bytes() | | 63 | } | | 64 |  | | 65 | *func* (*g* Gost) decrypt(*data*, *key* []*byte*, *mode* Gost\_Mode) []*byte* { | | 66 | switch mode { | | 67 | case GOST\_REPLACEMENT: | | 68 | openData := new(bytes.Buffer) | | 69 | for i := 0; i < len(data); i+=8 { | | 70 | blockBytes := data[i : i+8] | | 71 | block := g.decryptionCycle(g.bytesToUint64(blockBytes), key) | | 72 | openData.Write(g.uint64ToBytes(block)) | | 73 | } | | 74 | return openData.Bytes() | | 75 | default: | | 76 | fallthrough | | 77 | case GOST\_XORING: | | 78 | return g.encrypt(data, key, mode) | | 79 | } | | 80 | } | | 81 |  | | 82 | *func* (*g* Gost) uint64ToBytes(*block* *uint64*) []*byte* { | | 83 | bytes := make([]*byte*, 8, 8) | | 84 | binary.LittleEndian.PutUint64(bytes, block) | | 85 | return bytes | | 86 | } | | 87 |  | | 88 | *func* (*g* Gost) bytesToUint64(*bytes* []*byte*) *uint64* { | | 89 | return binary.LittleEndian.Uint64(bytes) | | 90 | } | | 91 |  | | 92 | *func* (*g* Gost) swap(*block* *uint64*) *uint64* { | | 93 | return ((block & 0xFFFFFFFF00000000) >> 32) | ((block & 0x00000000FFFFFFFF) << 32) | | 94 | } | | 95 |  | | 96 | *func* (*g* Gost) unionBlock(*part1*, *part2* *uint64*) *uint64* { | | 97 | return part1 << 32 | part2 | | 98 | } | | 99 |  | | 100 | *func* (*g* Gost) tableReplacement(*block* *uint64*) *uint64* { | | 101 | r := *uint64*(0) | | 102 | for i := 0; i < 8; i++ { | | 103 | index:= (block>>*uint*(4 \* i))&0x0F | | 104 | r |= *uint64*(g.table[i][index]) << *uint*(4 \* i) | | 105 | } | | 106 | return r | | 107 | } | | 108 |  | | 109 | *func* (*g* Gost) step(*block* *uint64*, *key* *byte*) *uint64* { | | 110 | part1 := block & 0x00000000FFFFFFFF | | 111 | part2 := block >> 32 | | 112 |  | | 113 | s := (part1 + *uint64*(key)) % math.MaxUint32 | | 114 | s = g.tableReplacement(s) | | 115 | s = s<<11 | s>>21 | | 116 | s = s ^ part2 | | 117 | return g.unionBlock(s, part1) | | 118 | } | | 119 |  | | 120 | *func* (*g* Gost) encryptionCycle(*block* *uint64*, *key* []*byte*) *uint64* { | | 121 | r:= block | | 122 | for k := 0; k < 3; k++ { | | 123 | for j := 0; j < len(key); j++ { | | 124 | r = g.step(r, key[j]) | | 125 | } | | 126 | } | | 127 | for j := len(key) - 1; j >= 0; j-- { | | 128 | r = g.step(r, key[j]) | | 129 | } | | 130 | return g.swap(r) | | 131 | } | | 132 |  | | 133 | *func* (*g* Gost) decryptionCycle(*block* *uint64*, *key* []*byte*) *uint64* { | | 134 | r:= block | | 135 | for j := 0; j < len(key); j++ { | | 136 | r = g.step(r, key[j]) | | 137 | } | | 138 | for k := 0; k < 3; k++ { | | 139 | for j := len(key) - 1; j >= 0; j-- { | | 140 | r = g.step(r, key[j]) | | 141 | } | | 142 | } | | 143 | return g.swap(r) | | 144 | } | | 145 |  | | 146 | *func* readKey(*reader* \*bufio.Reader, *size* *int*) *string* { | | 147 | fmt.Printf("Введите ключ длиной %d символов: ", size) | | 148 | value, \_ := reader.ReadString('\n') | | 149 | for len([]*byte*(strings.TrimSpace(value))) != size { | | 150 | fmt.Printf("Убедитесь, что длина ключа %d символов: ", size) | | 151 | tmp, \_ := reader.ReadString('\n') | | 152 | value = tmp | | 153 | } | | 154 | return value | | 155 | } | | 156 |  | | 157 | *func* main() { | | 158 | gost:= Gost{table: [8][16]*byte*{ | | 159 | {4, 10, 9, 2, 13, 8, 0, 14, 6, 11, 1, 12, 7, 15, 5, 3}, | | 160 | {14, 11, 4, 12, 6, 13, 15, 10, 2, 3, 8, 1, 0, 7, 5, 9}, | | 161 | {5, 8, 1, 13, 10, 3, 4, 2, 14, 15, 12, 7, 6, 0, 9, 11}, | | 162 | {7, 13, 10, 1, 0, 8, 9, 15, 14, 4, 6, 12, 11, 2, 5, 3}, | | 163 | {6, 12, 7, 1, 5, 15, 13, 8, 4, 10, 9, 14, 0, 3, 11, 2}, | | 164 | {4, 11, 10, 0, 7, 2, 1, 13, 3, 6, 8, 5, 9, 12, 15, 14}, | | 165 | {13, 11, 4, 1, 3, 15, 5, 9, 0, 10, 14, 7, 6, 8, 2, 12}, | | 166 | {1, 15, 13, 0, 5, 7, 10, 4, 9, 2, 3, 14, 6, 11, 8, 12}}} | | 167 |  | | 168 | reader := bufio.NewReader(os.Stdin) | | 169 | fmt.Print("Введите текст: ") | | 170 | text, \_ := reader.ReadString('\n') | | 171 | text = strings.TrimSpace(text) | | 172 |  | | 173 | key := readKey(reader, 32) | | 174 |  | | 175 | fmt.Print("Простая замена[1] или гаммирование[2]: ") | | 176 | mode, \_ := reader.ReadString('\n') | | 177 | for strings.TrimSpace(mode) != "1" && strings.TrimSpace(mode) != "2" { | | 178 | fmt.Print("Возможные вариант [1] или [2]: ") | | 179 | tmp, \_ := reader.ReadString('\n') | | 180 | mode = tmp | | 181 | } | | 182 |  | | 183 | fmt.Print("Зашифровать[1] или расшифровать[2]: ") | | 184 | direction, \_ := reader.ReadString('\n') | | 185 | for strings.TrimSpace(direction) != "1" && strings.TrimSpace(direction) != "2" { | | 186 | fmt.Print("Возможные вариант [1] или [2]: ") | | 187 | tmp, \_ := reader.ReadString('\n') | | 188 | direction = tmp | | 189 | } | | 190 |  | | 191 | modeId := GOST\_XORING | | 192 | switch strings.TrimSpace(mode) { | | 193 | case "1": | | 194 | modeId = GOST\_REPLACEMENT | | 195 | case "2": | | 196 | modeId = GOST\_XORING | | 197 | } | | 198 |  | | 199 | if strings.TrimSpace(direction) == "1" { | | 200 | fmt.Printf("Зашифрованный текст: %s\n", gost.encryptStr(text, key, modeId)) | | 201 | } else { | | 202 | fmt.Printf("Расшифрованный текст: %s\n", gost.decryptStr(text, key, modeId)) | | 203 | } | | 204 | } | |  |

1. **Результаты программы**

Результат кодирования и декодирования в режимах простой замены и гаммирования представлен на рисунке 2.1:

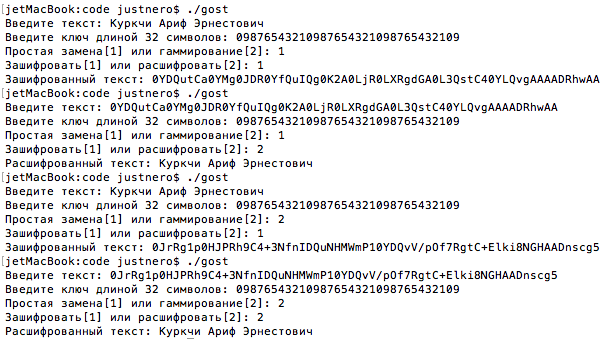


Рисунок 2.1 – Результаты в режимах простой замены и гаммирования

1. **Сравнительная характеристика DES и ГОСТ**

Краткая сравнительная характеристика двух представленных алгоритмов дана в таблице 3.1.

Таблица 3.1 – Краткая сравнительная характеристика ГОСТ и DES.

|  |  |  |
| --- | --- | --- |
| Параметр | ГОСТ | DES |
| Размер блока шифрования | 64 бита | 64 бита |
| Длина ключа | 256 бит | 56 бит |
| Число раундов |  |  |
| Узлы замен (S-блоки) | не фиксированы | фиксированы |
| Длина ключа для одного раунда | 32 бита | 48 бит |
| Схема выработки раундового ключа | простая | сложная |
| Начальная и конечная перестановки битов | нет | есть |

Функция шифрования ГОСТа гораздо проще функции шифрования DES, она не содержит операций битовых перестановок, коими изобилует DES и которые крайне неэффективно реализуются на современных универсальных процессорах. В силу сказанного, при вдвое большем количестве раундов (32 против 16) программная реализация ГОСТа на процессорах более чем в 2 раза превосходит по быстродействию реализацию DES. Естественно, сравнивались близкие к оптимуму по быстродействию реализации.

На каждом раунде шифрования используется "раундовый ключ", в DES он 48-битовый и вырабатывается по относительно сложному алгоритму, включающему битовые перестановки и замены по таблице, в ГОСТе он берется как фрагмент ключа шифрования. Длина ключа шифрования в ГОСТе равна 256 битам, длина раундового ключа - 32 битам, итого получаем, что ключ шифрования ГОСТа содержит 256/32=8 раундовых ключей. В ГОСТе 32 раунда, следовательно, каждый раундовый ключ используется 4 раза, порядок использования раундовых ключей установлен в ГОСТе и различен для различных режимов.

Таблица замен в ГОСТе – аналог S-блоков DES – представляет собой таблицу (матрицу) размером 8x16, содержащую число от 0 до 15. В каждой строке каждое из 16-ти чисел должно встретиться ровно 1 раз. В отличие от DES, таблица замен в ГОСТе одна и та же для всех раундов и не зафиксирована в стандарте, а является сменяемым секретным ключевым элементом.

В ГОСТе, в отличие от DES, нет начальной и конечной битовых перестановок шифруемого блока, которые, по мнению ряда специалистов, не влияют существенно на стойкость шифра, хотя влияют (в сторону уменьшения) на эффективность его реализации.

1. **Раунд AES**

Исходные данные для зашифровывания представлены в таблице 4.1:

Таблица 4.1 – Исходные данные

|  |  |  |  |
| --- | --- | --- | --- |
| к | у | р | к |
| ч | и | а | р |
| и | ф | э | р |
| н | е | с | т |

Исходные данные ключа для зашифровывания представлены в таблице 4.2:

Таблица 4.2 – Исходные данные ключа

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 6 | 1 | 0 |
| 1 | 9 | 9 | 7 |
| с | и | м | ф |
| е | р | о | п |

Исходные данные для зашифровывания в шестнадцатеричном формате представлены в таблице 4.3:

Таблица 4.3 – Исходные данные в шестнадцатеричном формате

|  |  |  |  |
| --- | --- | --- | --- |
| AA | E3 | E0 | AA |
| E7 | A8 | A0 | E0 |
| A8 | E4 | ED | E0 |
| AD | A5 | E1 | E2 |

Исходные данные ключа для зашифровывания в шестнадцатеричном формате представлены в таблице 4.4:

Таблица 4.4 – Исходные данные ключа в шестнадцатеричном формате

|  |  |  |  |
| --- | --- | --- | --- |
| 30 | 36 | 31 | 30 |
| 31 | 39 | 39 | 37 |
| E1 | A8 | AC | E4 |
| A5 | E0 | AE | AF |

1. Операция SubBytes. Байтом замены (таблица S-блоков) является узел пересечения старшей и младшей части байта. Например первый байт состояния Nb 88 заменяется на С4.

Результат операции SubBytes представлен в таблице 4.5:

Таблица 4.5 – Результат операции SubBytes

|  |  |  |  |
| --- | --- | --- | --- |
| AC | 11 | E1 | AC |
| 94 | C2 | E0 | E1 |
| C2 | 69 | 55 | E1 |
| 95 | 06 | F8 | 98 |

Результат данной операции в программе RijndaelDemo представлен на рисунке 4.1 ниже:

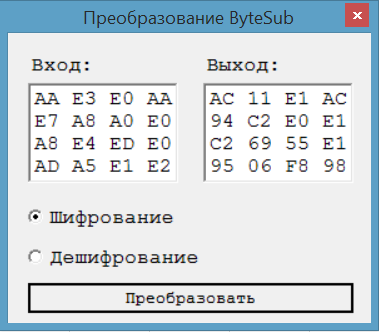


Рисунок 4.1 – Результат выполнения операции SubBytes программой RijndaelDemo

2. Операция ShiftRows. В стандарте шифрования AES в 0-ой строке состояния сдвиг не происходит, 1-ая строка сдвигается на 1 байт, 2-ая – на 2 байта, 3-я – на 3 байта.

Результат операции ShiftRows представлен в таблице 4.6:

Таблица 4.6 – Результат операции ShiftRows

|  |  |  |  |
| --- | --- | --- | --- |
| AC | 11 | E1 | AC |
| C2 | E0 | E1 | 94 |
| 55 | E1 | C2 | 69 |
| 98 | 95 | 06 | F8 |

Результат данной операции в программе RijndaelDemo представлен на рисунке 4.2 ниже:

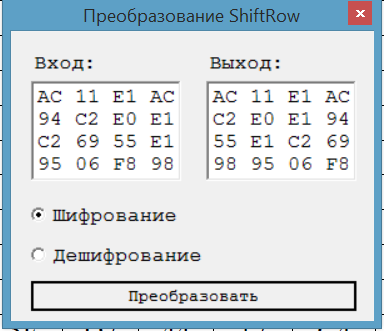


Рисунок 4.2 – Результат выполнения операции ShiftRows программой RijndaelDemo

3. Вычисление операции MixColumns представлено ниже:

Результат операции MixColumns представлен в таблице 4.7:

Таблица 4.7 – Результат операции MixColumns

|  |  |  |  |
| --- | --- | --- | --- |
| D3 | 6D | 25 | 75 |
| 54 | 67 | 63 | DC |
| 77 | 8C | 95 | F9 |
| 53 | 03 | 17 | F9 |

Результат операции в программе RijndaelDemo представлен на рисунке 4.3:

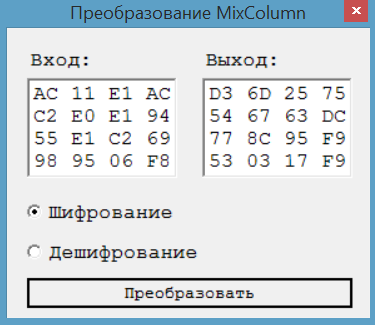


Рисунок 4.3 – Результат выполнения MixColumns программой RijndaelDemo

4. Операция AddRoundKey ключ раунда Round Key прибавляется к массиву State с помощью операции простого побитового сложения XOR (сложения по модулю 2).

Результат операции AddRoundKey представлен в таблице 4.8:

Таблица 4.8 – Результат операции AddRoundKey

|  |  |  |  |
| --- | --- | --- | --- |
| E3 | 5B | 14 | 45 |
| 65 | 5E | 5A | EB |
| 96 | 24 | 39 | 1D |
| F6 | E3 | B9 | 56 |

Результат данной операции в программе RijndaelDemo представлен на рисунке 4.4 ниже:

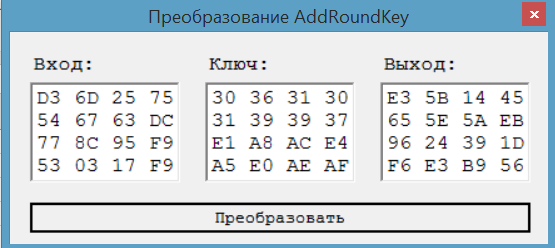


Рисунок 4.4 – Результат выполнения операции AddRoundKey программой RijndaelDemo

Как видно из рисунков и таблиц, собственные расчеты и расчеты, полученные с помощью программы, сошлись.

**Выводы**

В ходе данной лабораторной работы были изучены симметричные алгоритмы шифрования. Для этого были рассмотрены такие алгоритмы, как ГОСТ 28147-89, DES, AES.