|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Министерство образования и науки Российской Федерации | | | | | | | | | | |  |
| федеральное государственное бюджетное образовательное учреждение высшего образования «Российский государственный университет | | | | | | | | | | |  |
| им. А.Н. Косыгина (Технологии. Дизайн. Искусство)» | | | | | | | | | | |  |
| Институт информационных технологий и цифровой трансформации | | | | | | | | | | |  |
|  | | | | | | | | | | |  |
|  | | | | | | | | | | |  |
|  | | | | | | | | | | |  |
| КУРСОВАЯ РАБОТА  по дисциплине «Защита информации» | | | | | | | | | | |  |
| на тему: | разработка защищенного мессенджера, использующего алгоритмы защиты конфиденциальных данных: LOKI97, RC5 | | | | | | | | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | | | | | | | | | | |  |
| Направление подготовки: | | | | Искусственный интеллект, прикладная математика и программирование | | | | | | |  |
|  | | | |  | | | | | | |  |
|  | | | | | | | | | | | |
| Выполнил | | | | | | | | | | | |
| студент | | группы | МПМ-121 |  | 4 | курса | очной | | | формы обучения | |
|  | |  |  |  |  |  |  | | |  | |
|  | | | | |  | | |  | Красина Анжелика  Сергеевна | |  |
|  | | | | |  | | |  |  | |  |
|  | | | | |  | | |  |  | |  |
| Руководитель: | | | | |  | | |  | Ирбитский Илья  Сергеевич | |  |
|  | | |  |  | |  |
| Москва 2024  СОДЕРЖАНИЕ  Введение 3  1 Теоретическая часть 5  1.1 Алгоритм RC5  5  1.2 Алгоритм LOKI97 6  1.3 Протокол Диффи-Хеллмана 8  2 Практическая часть 18  2.1. Архитектура комплекса приложений 18  2.2 Использованные при разработке технологии 19  Заключение 23  Список использованных источников и литературы 24 | | | | | | | | | | |  |

ВВЕДЕНИЕ

Современные системы информационной безопасности требуют применения эффективных и надежных методов шифрования для защиты данных от несанкционированного доступа. В условиях стремительного роста числа киберугроз и увеличения объема информации, передаваемой через цифровые каналы, разработка приложений с интеграцией алгоритмов шифрования становится актуальной задачей.

Целью данной курсовой работы является разработка программного комплекса, включающего серверное приложение и клиентский интерфейс, реализующего защищенный обмен данными между пользователями с использованием алгоритмов симметричного шифрования RC5 и MacGuffin, а также протокола Диффи-Хеллмана для генерации сеансовых ключей.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Реализовать два алгоритма симметричного шифрования, соответствующих стандартным интерфейсам, без использования сторонних библиотек.
2. Реализовать протокол Диффи-Хеллмана для генерации и распределения сеансовых ключей.
3. Создать серверное приложение с поддержкой асинхронной передачи данных через стриминг и сетевые протоколы, обеспечивающее управление секретными чатами.
4. Разработать клиентское приложение с интуитивным пользовательским интерфейсом для подключения к секретным чатам, выполнения операций шифрования и дешифрования, а также управления передачей зашифрованных данных.
5. Обеспечить совместимость серверного приложения с инфраструктурой контейнеризации Docker для облегчения развертывания и масштабирования.

Актуальность данной работы обусловлена необходимостью разработки приложений, предоставляющих пользователям возможность безопасного обмена данными в условиях ограничений вычислительных ресурсов и высокой вероятности атак на каналы передачи информации. Выбор алгоритмов RC5 и MacGuffin основан на их теоретической основе и практической применимости для задач симметричного шифрования.

Результаты данной работы могут быть использованы для создания решений в области защиты информации, включая корпоративные коммуникации, электронный документооборот и конфиденциальное общение в цифровых средах.

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Алгоритм RC5

Алгоритм RC5 был представлен Роном Ривестом, одним из основателей RSA Security, в рамках исследований по разработке быстрых и безопасных симметричных шифров. RC5 был разработан как простой, но мощный алгоритм, способный адаптироваться к различным требованиям безопасности и производительности. Его гибкость выражается в возможности изменения размера блока, длины ключа и числа раундов шифрования, что позволяет оптимизировать алгоритм под конкретные задачи и ограничения. Со временем были предложены различные вариации RC5, такие как RC5-32/12/16 (где 32 — размер слова в битах, 12 — количество раундов, 16 — длина ключа в байтах) и другие конфигурации, адаптированные под разные платформы и требования безопасности. Несмотря на появление новых алгоритмов, RC5 продолжает оставаться актуальным благодаря своей универсальности и эффективности.

RC5 относится к классу блочных шифров и использует схему Фейстеля для обеспечения безопасности. Основные параметры RC5 включают:

* **w**: размер слова в битах (16, 32 или 64 бита)
* **r**: количество раундов (обычно от 12 до 20)
* **b**: длина ключа в байтах (до 255 байт)

Таким образом, полная запись алгоритма RC5 включает три параметра: RC5‑w/r/b.RC5 основан на трех основных операциях:

**1. Сложение по модулю 2** : Обеспечивает нелинейность и усложняет криптоанализ.

**2. Исключающее ИЛИ (XOR)**: Вводит элемент случайности и способствует диффузии.

**3. Циклический сдвиг (ROTL, ROTR)**: Создает сложные взаимосвязи между битами, усиливая диффузию.

Эти операции комбинируются для создания сложной и трудно предсказуемой структуры, обеспечивающей высокую степень безопасности.

Процесс расширения ключа в RC5 преобразует исходный ключ произвольной длины в массив под-ключей, используемых в процессе шифрования и дешифрования. Этот процесс состоит из нескольких этапов:

1. Инициализация массива S:

Массив S содержит под-ключи, используемые на каждом раунде шифрования и дешифрования. Размер массива определяется как 2(r+1), где r — количество раундов. Первый этап расширения ключа включает инициализацию S значениями:

S[0]=Pw​

S[i]=S[i−1]+Qw​, для i=1,2,…,2(r+1)−1

где:

* + - Pw​=Odd((e−2)×2w)
    - Qw​=Odd((ϕ−1)×2w)
    - e — основание натурального логарифма
    - ϕ — золотое сечение (ϕ=(1+sqrt 5)/2)
    - Odd(x) — ближайшее нечетное число к х.

Эти константы Pw​ и Qw​ выбраны таким образом, чтобы их двоичное представление не содержало повторяющихся паттернов, что способствует лучшему смешиванию ключа.

1. Копирование и подготовка ключа K:

Исходный ключ K длиной b байт преобразуется в массив L из c=⌈8b/w⌉ слов длиной w бит. Если длина ключа K не кратна размеру слова w, она дополняется нулями до ближайшего слова.

1. Перемешивание массива S и массива L:

Массивы S и L перемешиваются с использованием циклического сдвига и сложения по модулю 2<sup>w</sup>. Этот процесс повторяется 3 × max(c, t) раз, где t=2(r+1).

В каждой итерации выполняются следующие операции:

S[i]=(S[i]+A+B)⋘3

L[j]=(L[j]+A+B)⋘(A+B)

где:

* + A=S[i]
  + B=L[j]
  + i=(i+1)modt
  + j=(j+1)modc

Циклический сдвиг осуществляется на количество бит, равное значению суммы A+B. Этот процесс обеспечивает тщательное смешивание исходного ключа с массивом S, создавая уникальные под-ключи для каждого раунда шифрования

Шифрование в RC5 выполняется над блоками данных фиксированного размера (2w бит). Рассмотрим процесс шифрования шаг за шагом:

1. Инициализация блоков данных:

Данные разбиваются на блоки размером 2w бит. Каждый блок разделяется на два регистра:

A,B←Разделить блок на два слова по w бит

1. Начальная операция с под-ключами:

К каждому регистру добавляется соответствующий элемент массива S: A=A+S[0]

B=B+S[1]

1. Раунды шифрования:

Для каждого раунда i от 1 до r выполняются следующие операции:

A=((A⊕B)⋘B)+S[2i]

B=((B⊕A)⋘A)+S[2i+1]

где:

* + ⊕ — операция XOR
  + ⋘ — циклический сдвиг влево на количество бит, равное значению регистра

Эти операции обеспечивают нелинейное смешивание регистров A и B, что затрудняет анализ шифра и повышает его устойчивость к криптоатакам.

1. Формирование зашифрованного блока:

После выполнения всех раундов шифрования, зашифрованный блок формируется путем объединения регистров A и B:

Зашифрованный блок=(A∥B)

Дешифрование в RC5 выполняется как обратный процесс шифрования. Основная цель — восстановить исходные данные из зашифрованного блока, используя тот же расширенный ключ S:

1. Инициализация зашифрованного блока:

Зашифрованный блок разбивается на два регистра:

A,B←Разделить зашифрованный блок на два слова по w бит

1. Обратные раунды шифрования:

Дешифрование начинается с последнего раунда и проходит в обратном порядке до первого. Для каждого раунда i от r до 1 выполняются следующие операции:

B=((B−S[2i+1])⋙A)⊕A

A=((A−S[2i])⋙B)⊕B

где:

* + - ⋙ — циклический сдвиг вправо на количество бит, равное значению регистра
    - − — вычитание по модулю 2<sup>w</sup>

Эти операции эффективно отменяют действия, выполненные на этапе шифрования, восстанавливая значения регистров A и B до их состояния после начальной операции.

1. Обратная начальная операция с под-ключами:

После выполнения всех обратных раундов, к каждому регистру вычитаются соответствующие элементы массива S:

A=A−S[0]

B=B−S[1]

1. Восстановление исходного блока:

Полученные значения регистров A и B объединяются, формируя исходный (расшифрованный) блок данных:

Исходный блок=(A∥B)

1.2 Алгоритм LOKI97

**LOKI97** является частью семейства блочных шифров **LOKI**, разработанных группой исследователей в компании **IBM**. Первоначально был представлен алгоритм **LOKI89**, затем **LOKI91**, и, наконец, **LOKI97**, который стал одной из наиболее известных и изученных версий данного семейства. Целью разработки LOKI97 было создание блочного шифра, обладающего высокой степенью безопасности и устойчивого к известным криптоатакам на момент его создания. LOKI97 был представлен в 1997 году и привлек внимание криптографов своей простой и элегантной структурой, основанной на перестановках и замещениях, что напоминает структуру алгоритма DES (Data Encryption Standard). Несмотря на то, что DES уже в то время считался устаревшим из-за относительно короткой длины ключа, LOKI97 предложил ряд улучшений, направленных на повышение безопасности и эффективности.

## Основные параметры алгоритма

* **Структура**: Feistel
* **Количество раундов**: 16
* **Размер блока данных**: 128 бит (16 байт)
* **Длина ключа**: 128, 192 или 256 бит
* **Ключевое расписание**: 256-битное, состоящее из 48 под-ключей (по 3 под-ключа на раунд)

**LOKI97** относится к классу симметричных блочных шифров, использующих структуру Feistel. Основные компоненты алгоритма включают:

1. **Структура Feistel**: Блок данных разделяется на две равные части, каждая из которых последовательно обрабатывается в раундах шифрования с использованием под-ключей.
2. **Функция замещения (S-блоки)**: Нелинейные преобразования, обеспечивающие криптоустойчивость алгоритма.
3. **Функция перестановки (P-блоки)**: Перестановки битов для обеспечения диффузии данных.
4. **Ключевое расписание**: Процесс генерации под-ключей из основного ключа для каждого раунда шифрования и дешифрования.

Ключевое расписание в **LOKI97** играет важную роль в обеспечении безопасности алгоритма, отвечая за генерацию под-ключей, используемых на каждом раунде шифрования и дешифрования. Процесс ключевого расписания включает следующие этапы:

### 1. Инициализация

Основной ключ длиной 128, 192 или 256 бит разбивается на четыре 64-битных слова: K40​, K30​, K20​, K10​. В зависимости от длины ключа, некоторые слова получают значения, полученные с помощью функции f:

* **128-битный ключ [Ka | Kb]**:

[K40​∥K30​∥K20​∥K10​]=[Ka∥Kb∥f(Kb,Ka)∥f(Ka,Kb)]

* **192-битный ключ [Ka | Kb | Kc]**:

[K40​∥K30​∥K20​∥K10​]=[Ka∥Kb∥Kc∥f(Ka,Kb)]

* **256-битный ключ [Ka | Kb | Kc | Kd]**:

[K40​∥K30​∥K20​∥K10​]=[Ka∥Kb∥Kc∥Kd]

### 2. Генерация под-ключей

Процесс генерации под-ключей включает 48 раундов, по три под-ключа на каждый раунд:

SKi​=K1i​=K4i−1​⊕g(K1i−1​,K3i−1​,K2i−1​) K4i​=K3i−1​ K3i​=K2i−1​ K2i​=K1i−1​

где функция g определяется как:

g(K1​,K3​,K2​)=f(K1​+K3​+(Δ×i),K2​)

**Delta** (Δ) — магическая константа, равная 0x9E3779B97F4A7C1516, полученная из золотого сечения. Эта константа используется для устранения симметрии в ключевом расписании и обеспечения непредсказуемости под-ключей.

### 3. Итоговое ключевое расписание

Таким образом, для каждого раунда шифрования генерируются три под-ключа, что обеспечивает общую безопасность и непредсказуемость алгоритма. В результате получается 48 под-ключей для 16 раундов шифрования.

## Процесс шифрования

Процесс шифрования в **LOKI97** основан на структуре Feistel и включает 16 раундов обработки данных. Рассмотрим основные этапы шифрования:

### 1. Разделение блока данных

128-битный блок данных разделяется на две равные части по 64 бита каждая:

* Левая часть (L0​)
* Правая часть (R0​)

### 2. Начальное смешивание с под-ключами

Перед началом раундов выполняется начальное смешивание с первыми двумя под-ключами:

L=(L0​+SK1​)mod264 R=(R0​+SK2​)mod264

### 3. Раунды шифрования

В каждом из 16 раундов выполняются следующие операции:

Li​=Ri−1​⊕f(Li−1​+SK3i−2​,SK3i−1​) Ri​=(Li−1​+SK3i​)mod264

где f — сложная нелинейная функция, обеспечивающая криптоустойчивость алгоритма.

### 4. Финальное смешивание с под-ключами

После завершения всех раундов выполняется финальное смешивание с под-ключами:

L16​=(L16​+SK49​)mod264 R16​=(R16​+SK50​)mod264

### 5. Формирование зашифрованного блока

Зашифрованный блок формируется путём объединения изменённых половин:

Ciphertext=[R16​∥L16​]

## Процесс дешифрования

Дешифрование в **LOKI97** является обратным процессом шифрования и также основано на структуре Feistel. Основная цель — восстановить исходные данные из зашифрованного блока с использованием тех же под-ключей, что и при шифровании.

### 1. Разделение зашифрованного блока

Зашифрованный 128-битный блок данных разделяется на две равные части по 64 бита каждая:

* Левая часть (L16​)
* Правая часть (R16​)

### 2. Обратное смешивание с под-ключами

Перед началом раундов выполняется обратное начальное смешивание с последними двумя под-ключами:

L=(L16​−SK49​)mod264 R=(R16​−SK50​)mod264

### 3. Раунды дешифрования

В каждом из 16 раундов выполняются обратные операции:

Ri−1​=Li​⊕f(Ri​−SK3i​,SK3i−1​) Li−1​=(Ri​−SK3i+1​)mod264

### 4. Финальное обратное смешивание с под-ключами

После завершения всех раундов выполняется финальное обратное смешивание с под-ключами:

L0​=(L0​−SK1​)mod264 R0​=(R0​−SK2​)mod264

### 5. Восстановление исходного блока данных

Восстановленный блок данных формируется путём объединения изменённых половин:

Plaintext=[L0​∥R0​]

## Функция f(A,B)

Функция f(A,B) является ключевым компонентом алгоритма **LOKI97**, обеспечивая высокую степень нелинейности и криптоустойчивости. Она состоит из следующих этапов:

f(A,B)=Sb(P(Sa(E(KP(A,B)))),B)

### Этапы выполнения функции f(A,B)

1. **Ключевая перестановка (KP)**:

 битов использованием под-ключейKP(A,B)=перестановка битовAиBс использованием под-ключей

* + Выполняется обмен битами между A и B в зависимости от под-ключей, обеспечивая начальную диффузию данных.

1. **Дополнительная перестановка (E)**:

 перестановка битов, обеспечивающая диффузиюE(KP(A,B))=специфическая перестановка битов, обеспечивающая диффузию

* + Выполняется фиксированная перестановка битов для дальнейшей диффузии данных.

1. **Первый слой S-блоков (Sa)**:

 первого слоя S-блоковSa(E(KP(A,B)))=применение первого слоя S-блоков

* + Включает применение двух различных S-блоков для обеспечения нелинейности.

1. **Перестановка (P)**:

 битов для максимальной диффузииP(Sa(E(KP(A,B))))=перестановка битов для максимальной диффузии

* + Выполняется дополнительная перестановка битов для обеспечения максимальной диффузии данных.

1. **Второй слой S-блоков (Sb)**:

 второго слоя S-блоков с учётом Sb(P(Sa(E(KP(A,B)))),B)=применение второго слоя S-блоков с учётом B

* + Повторное применение S-блоков для усиления нелинейности и криптоустойчивости.

Таким образом, функция f(A,B) обеспечивает сложные нелинейные преобразования, затрудняющие криптоанализ и обеспечивающие высокую степень криптоустойчивости алгоритма.

## Операции в поле Галуа

Для обеспечения высокой степени нелинейности в функции f(A,B), **LOKI97** использует операции в полях Галуа GF(2n). В частности, используются следующие операции:

### Умножение в поле Галуа

Функция galoisMultiply(a,b,mod,bits) реализует умножение двух элементов a и b в поле Галуа с порождающим полиномом mod и заданным количеством битов bits:

galoisMultiply(a,b,mod,bits)=a×bmod(mod)

**Описание**:

* Начинается с инициализации результата нулём.
* Последовательно проверяет каждый бит b. Если бит равен 1, добавляет текущий a к результату.
* Сдвигает a влево на один бит и, при необходимости, выполняет модульное сокращение с использованием порождающего полинома mod.
* Продолжает процесс до обработки всех битов b.

### Возведение в куб в поле Галуа

Функция galoisCube(x,mod,bits) реализует возведение элемента x в куб в поле Галуа:

galoisCube(x,mod,bits)=(x2×x)mod(mod)

**Описание**:

* Сначала вычисляется квадрат x2 с использованием функции умножения.
* Затем результат умножается на x, снова используя функцию умножения.
* Полученный результат представляет собой x3 в поле GF(2n).

### S-блоки: S1 и S2

**S1** и **S2** являются основными компонентами S-блоков, выполняющими нелинейные преобразования:

S1(x)=byte(x3modGF(213)&0xFF) S2(x)=byte(x3modGF(211)&0xFF)

**Описание**:

* **S1**:
  + Инвертирует входной 16-битный элемент и ограничивает его до 13 бит.
  + Возводит результат в куб в поле GF(213) с порождающим полиномом 0x2911.
  + Маскирует результат до 8 младших бит.
* **S2**:
  + Инвертирует входной 16-битный элемент и ограничивает его до 11 бит.
  + Возводит результат в куб в поле GF(211) с порождающим полиномом 0xA7.
  + Маскирует результат до 8 младших бит.

Эти преобразования обеспечивают высокую степень нелинейности и криптоустойчивости S-блоков, затрудняя криптоанализ и обеспечивая стойкость алгоритма к различным атакам.

## Функции перестановки: KP, E, P

### Ключевая перестановка (KP)

Функция **KP(A, B)** выполняет ключевую перестановку, основанную на битовых операциях между двумя 64-битными словами A и B:

 битов использованием под-ключейKP(A,B)=перестановка битовAиBс использованием под-ключей

**Описание**:

* Разделяет A и B на левые (Al) и правые (Ar) половины.
* Для каждого бита под-ключа SKr, если бит равен 1, соответствующие биты в Al и Ar меняются местами.
* Обеспечивает начальную диффузию данных перед применением S-блоков.

### Дополнительная перестановка (E)

Функция **E(input)** выполняет дополнительную перестановку битов входного 64-битного слова, преобразуя его в 96-битный результат:

 перестановка битов, обеспечивающая диффузиюE(input)=специфическая перестановка битов, обеспечивающая диффузию

**Описание**:

* Выполняет фиксированную перестановку битов для дальнейшей диффузии данных перед применением S-блоков.
* Обеспечивает рассеивание информации по всему блоку данных.

### Перестановка битов (P)

Функция **P(input)** выполняет перестановку битов входного 64-битного слова согласно заданной таблице перестановок:

 битов для максимальной диффузииP(input)=перестановка битов для максимальной диффузии

**Описание**:

* Использует фиксированную таблицу перестановок для перемещения битов, обеспечивая максимальную диффузию данных.
* Усиливает распространение изменений во входных данных по всему блоку.

## Алгоритм шифрования и дешифрования

### Шифрование

1. **Разделение блока данных**:
   * 128-битный блок данных разделяется на две равные части по 64 бита: левая (L0​) и правая (R0​).
2. **Начальное смешивание**:

L=(L0​+SK1​)mod264 R=(R0​+SK2​)mod264

1. **16 раундов шифрования**: В каждом раунде выполняются следующие операции:

Li​=Ri−1​⊕f(Li−1​+SK3i−2​,SK3i−1​) Ri​=(Li−1​+SK3i​)mod264

* + **f(A,B)**: Высоконелинейная функция, обеспечивающая криптоустойчивость алгоритма.

1. **Финальное смешивание**:

L16​=(L16​+SK49​)mod264 R16​=(R16​+SK50​)mod264

1. **Формирование зашифрованного блока**:

Ciphertext=[R16​∥L16​]

### Дешифрование

Дешифрование является обратным процессом шифрования и включает следующие этапы:

1. **Разделение зашифрованного блока**:
   * Зашифрованный 128-битный блок данных разделяется на две равные части по 64 бита: левая (L16​) и правая (R16​).
2. **Обратное смешивание**:

L=(L16​−SK49​)mod264 R=(R16​−SK50​)mod264

1. **16 раундов дешифрования**: В каждом раунде выполняются обратные операции:

Ri−1​=Li​⊕f(Ri​−SK3i​,SK3i−1​) Li−1​=(Ri​−SK3i+1​)mod264

1. **Финальное обратное смешивание**:

L0​=(L0​−SK1​)mod264 R0​=(R0​−SK2​)mod264

1. **Восстановление исходного блока данных**:

Plaintext=[L0​∥R0​]

## Функция f(A,B)

Функция f(A,B) является основным нелинейным компонентом алгоритма **LOKI97**, обеспечивающим криптоустойчивость и максимальную диффузию данных. Она состоит из следующих этапов:

f(A,B)=Sb(P(Sa(E(KP(A,B)))),B)

### Этапы выполнения функции f(A,B)

1. **Ключевая перестановка (KP)**:

 битов использованием под-ключейKP(A,B)=перестановка битовAиBс использованием под-ключей

* + Выполняется обмен битами между A и B в зависимости от под-ключей, обеспечивая начальную диффузию данных.

1. **Дополнительная перестановка (E)**:

 перестановка битов, обеспечивающая диффузиюE(KP(A,B))=специфическая перестановка битов, обеспечивающая диффузию

* + Выполняется фиксированная перестановка битов для дальнейшей диффузии данных перед применением S-блоков.

1. **Первый слой S-блоков (Sa)**:

 первого слоя S-блоковSa(E(KP(A,B)))=применение первого слоя S-блоков

* + Включает применение двух различных S-блоков для обеспечения нелинейности.

1. **Перестановка (P)**:

 битов для максимальной диффузииP(Sa(E(KP(A,B))))=перестановка битов для максимальной диффузии

* + Выполняется дополнительная перестановка битов для обеспечения максимальной диффузии данных.

1. **Второй слой S-блоков (Sb)**:

 второго слоя S-блоков с учётом Sb(P(Sa(E(KP(A,B)))),B)=применение второго слоя S-блоков с учётом B

* + Повторное применение S-блоков для усиления нелинейности и криптоустойчивости.

Таким образом, функция f(A,B) обеспечивает сложные нелинейные преобразования, затрудняющие криптоанализ и обеспечивающие высокую степень криптоустойчивости алгоритма.

## Операции в поле Галуа

Для обеспечения высокой степени нелинейности в функции f(A,B), **LOKI97** использует операции в полях Галуа GF(2n). В частности, используются следующие операции:

### Умножение в поле Галуа

Функция **умножение в поле Галуа** реализует умножение двух элементов a и b в поле Галуа с порождающим полиномом mod и заданным количеством битов bits:

galoisMultiply(a,b,mod,bits)=a×bmod(mod)

**Описание**:

* Начинается с инициализации результата нулём.
* Последовательно проверяет каждый бит b. Если бит равен 1, добавляет текущий a к результату.
* Сдвигает a влево на один бит и, при необходимости, выполняет модульное сокращение с использованием порождающего полинома mod.
* Продолжает процесс до обработки всех битов b.

### Возведение в куб в поле Галуа

Функция **возведение в куб** реализует возведение элемента x в куб в поле Галуа:

galoisCube(x,mod,bits)=(x2×x)mod(mod)

**Описание**:

* Сначала вычисляется квадрат x2 с использованием функции умножения.
* Затем результат умножается на x, снова используя функцию умножения.
* Полученный результат представляет собой x3 в поле GF(2n).

### S-блоки: S1 и S2

**S1** и **S2** являются основными компонентами S-блоков, выполняющими нелинейные преобразования:

S1(x)=byte(x3modGF(213)&0xFF) S2(x)=byte(x3modGF(211)&0xFF)

**Описание**:

* **S1**:
  + Инвертирует входной 16-битный элемент и ограничивает его до 13 бит.
  + Возводит результат в куб в поле GF(213) с порождающим полиномом 0x2911.
  + Маскирует результат до 8 младших бит.
* **S2**:
  + Инвертирует входной 16-битный элемент и ограничивает его до 11 бит.
  + Возводит результат в куб в поле GF(211) с порождающим полиномом 0xA7.
  + Маскирует результат до 8 младших бит.

Эти преобразования обеспечивают высокую степень нелинейности и криптоустойчивости S-блоков, затрудняя криптоанализ и обеспечивая стойкость алгоритма к различным атакам.

## Функции перестановки: KP, E, P

### Ключевая перестановка (KP)

Функция **KP(A, B)** выполняет ключевую перестановку, основанную на битовых операциях между двумя 64-битными словами A и B:

 битов использованием под-ключейKP(A,B)=перестановка битовAиBс использованием под-ключей

**Описание**:

* Разделяет A и B на левые (Al) и правые (Ar) половины.
* Для каждого бита под-ключа SKr, если бит равен 1, соответствующие биты в Al и Ar меняются местами.
* Обеспечивает начальную диффузию данных перед применением S-блоков.

### Дополнительная перестановка (E)

Функция **E(input)** выполняет дополнительную перестановку битов входного 64-битного слова, преобразуя его в 96-битный результат:

 перестановка битов, обеспечивающая диффузиюE(input)=специфическая перестановка битов, обеспечивающая диффузию

**Описание**:

* Выполняет фиксированную перестановку битов для дальнейшей диффузии данных перед применением S-блоков.
* Обеспечивает рассеивание информации по всему блоку данных.

### Перестановка битов (P)

Функция **P(input)** выполняет перестановку битов входного 64-битного слова согласно заданной таблице перестановок:

 битов для максимальной диффузииP(input)=перестановка битов для максимальной диффузии

**Описание**:

* Использует фиксированную таблицу перестановок для перемещения битов, обеспечивая максимальную диффузию данных.
* Усиливает распространение изменений во входных данных по всему блоку.

## Безопасность алгоритма LOKI97

### Сильные стороны

1. **Структура Feistel**: Обеспечивает симметричность алгоритма и упрощает процессы шифрования и дешифрования.
2. **Нелинейные операции**: Использование S-блоков и P-блоков вводит высокую степень нелинейности, усложняя криптоанализ.
3. **Длина ключа**: Поддержка ключей длиной до 256 бит обеспечивает высокий уровень безопасности против атак грубой силы.
4. **Магическая константа Delta**: Использование порождающей константы, полученной из золотого сечения, помогает устранить симметрию в ключевом расписании.
5. **Многократные раунды**: 16 раундов шифрования обеспечивают достаточную диффузию и смешивание данных, повышая устойчивость к криптоатакам.

1.3 Протокол Диффи-Хеллмана

До появления алгоритма Диффи-Хеллмана, основные методы обеспечения безопасности передавались посредством **симметричных ключей**, где одна и та же секретная информация использовалась как для шифрования, так и для дешифрования сообщений. Однако возникла проблема безопасной передачи этих ключей между сторонами. Решение этой проблемы предложили Диффи и Хеллман, представив метод обмена ключами, который позволяет двум сторонам установить общий секретный ключ без предварительного обмена секретной информацией.

Алгоритм Диффи-Хеллмана стал основой для последующих разработок в области публичных ключевых криптосистем, таких как RSA и эллиптические кривые. Его значимость подтверждается тем, что он заложил фундаментальные принципы, используемые в современных протоколах безопасности, включая SSL/TLS для защиты интернет-соединений.

### Публичные и приватные ключи

В контексте алгоритма Диффи-Хеллмана, каждая сторона генерирует **приватный ключ** (секретное число) и **публичный ключ** (число, полученное путём математических преобразований приватного ключа). Публичные ключи могут свободно передаваться по открытым каналам связи, тогда как приватные ключи остаются в тайне.

### Математическая основа: Дискретный логарифм

Безопасность алгоритма Диффи-Хеллмана основывается на сложности задачи **дискретного логарифма**. В математике, для заданного основания g и числа h в группе G, задача дискретного логарифма заключается в нахождении целого числа x, такого что:

h=gxmodp

где p — большое простое число, определяющее размер группы.

Решение этой задачи является вычислительно трудной задачей, особенно при использовании больших простых чисел, что обеспечивает криптографическую стойкость алгоритма.

## Принцип работы алгоритма Диффи-Хеллмана

Алгоритм Диффи-Хеллмана позволяет двум сторонам (например, Алисе и Бобу) безопасно установить общий секретный ключ, используя публичный канал связи. Процесс можно описать следующими шагами:

### 1. Выбор общих параметров

* **Простое число p**: Алиса и Боб согласовывают большое простое число p, которое служит модулем для всех вычислений.
* **Основание g**: Они выбирают целое число g, такое что 1<g<p, и g является генератором мультипликативной группы Zp∗​.

Эти параметры p и g могут быть общедоступными и не требуют секретности.

### 2. Генерация приватных и публичных ключей

* **Алиса**:
  + Выбирает приватный ключ a, случайное целое число в диапазоне 1<a<p−1.
  + Вычисляет публичный ключ A=gamodp.
* **Боб**:
  + Выбирает приватный ключ b, случайное целое число в диапазоне 1<b<p−1.
  + Вычисляет публичный ключ B=gbmodp.

### 3. Обмен публичными ключами

Алиса отправляет Бобу свой публичный ключ A, а Боб отправляет Алисе свой публичный ключ B. Эти публичные ключи могут быть переданы по открытому каналу связи без риска раскрытия приватных ключей.

### 4. Вычисление общего секретного ключа

* **Алиса**:
  + Принимает публичный ключ B от Боба.
  + Вычисляет общий секретный ключ s=Bamodp.
* **Боб**:
  + Принимает публичный ключ A от Алисы.
  + Вычисляет общий секретный ключ s=Abmodp.

### 5. Результат

Обе стороны, Алиса и Боб, получат одинаковый общий секретный ключ s, поскольку:

s=Bamodp=(gbmodp)amodp=gbamodp=gabmodp=(gamodp)bmodp=Abmodp

Этот общий секретный ключ s может быть использован для симметричного шифрования дальнейшей коммуникации, обеспечивая её конфиденциальность.

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1. Архитектура комплекса приложений

2.2 Использованные при разработке технологии

ЗАКЛЮЧЕНИЕ

В результате работы было получено приложение, которое представляет собой

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Донован А. А., Керниган Б.У. Язык программирования Go / пер. с англ. М.: Вильямс, 2018. 432с.
2. Клеппман М. Высоконагруженные приложения. Программирование, масштабирование, поддержка / пер. с англ. С-Пб.: Питер, 2024. 640с.
3. Смарт Н. Криптография. – М.: Техносфера, 2005. -525 с.
4. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. – М.:Триумф, 2012. -815 с.
5. Цукалос М. Golang для профи. Работа с сетью, многопоточность, структуры данных и машинное обучение с Go. - С-Пб.: Питер, 2021. 720с.