МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)» (МАИ)

Институт №8 «Информационные технологии и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

КУРСОВОЙ ПРОЕКТ

по дисциплине «Основы криптографии»

на тему: «Разработка приложения для обмена данными с шифрованием алгоритмами RC5 и RC6»

Студент: Сарафанников Я.В.

Группа: М8О-311Б-21

Подпись _____

Руководитель: Романенков А.М.

Подпись _____

Оценка:

Дата:

Москва, 2024

Содержание

1. Введение 3
1.1 Задание к курсовому проекту 3
2. Теоретическая часть 6
2.1 RC5 6
2.1.1 Параметры 6
2.1.2 Расширение ключа 6
2.1.2.1 Генерация констант 6
2.1.2.2 Разбиение ключа на слова 7
2.1.2.3 Построение таблицы расширенных ключей 7
2.1.2.4 Перемешивание 7
2.1.3 Шифрование 8
2.1.4 Расшифрование 8
2.2 RC6 9
2.2.1 Параметры 9
2.2.2 Расширение ключа 9
2.2.2.1 Генерация констант 9
2.2.2.2 Построение таблицы расширенных ключей 10
2.2.3 Шифрование 10
2.2.4 Расшифрование 11
3. Практическая часть 12
3.1 Архитектура приложения 12
3.1.1 Сервер 12
3.1.2 Клиент 17
3.1.3 Библиотека-модуль шифрования 19
3.2 Описание взаимодействий 20
3.2.1 Синхронизация клиентов, обмен ключами 20
3.2.2 Общение через брокеров Apache Kafka 22
3.3 Используемые технологии 22
4. Вывод 23
5. Список используемых источников 24
6. Приложение 25

1. Введение

1.1 Задание к курсовому проекту

1. Реализовать два симметричных алгоритма шифрования, реализующих интерфейс симметричного алгоритма шифрования (использование сторонних реализаций алгоритмов не допускается);
2. Реализовать протокол Диффи-Хеллмана (использование сторонних реализаций протокола не допускается);
3. Реализовать консольное серверное приложение, взаимодействие с которым возможно при помощи сетевого протокола и надстроек над ним (REST API, gRPC, WCF, etc.), API которого предусматривает возможность выполнения следующих протоколов (протокол может быть организован посредством выполнения нескольких связанных запросов):
    - Создание “комнаты” для организации защищённого обмена сообщениями между двумя клиентами (секретный чат) с возможностью выбора используемого алгоритма шифрования из реализованных в п. 1;
    - Закрытие ранее созданного секретного чата;
    - Подключение и отключение клиентов от секретного чата;
    - Организация распределения сеансового ключа симметричного алгоритма, сгенерированного посредством протокола Диффи-Хеллмана;
    - Обработка пользовательских запросов на приём и передачу данных между клиентами, участвующими в защищённом обмене сообщениями.

Взаимодействие с сервером должно быть реализовано в асинхронном стиле с использованием брокера сообщений (Redpanda, Apache Kafka, Apache ActiveMQ Artemis, RabbitMQ, MSMQ и т. д.), предоставляющего очередь сообщений, producer и consumer которой находятся на стороне серверного приложения. Сообщениями, поступающими в брокер, должны являться фрагменты шифротекста. После передачи зашифрованного фрагмента сообщения клиенту-потребителю запрещено дальнейшее *контролируемое* хранение этого фрагмента. Формат сериализации сообщений продумайте самостоятельно.

Опционально: реализовать этап развёртывания серверного приложения на основе абстракции инверсии управления (IoC), реализованной в виде механизма внедрения зависимости (DI). Конфигурирование этапа

развёртывания обеспечьте при помощи конфигурационных файлов и их трансформаций.

Опционально: обеспечить возможность репликации хранящихся в брокере сообщений данных.

4. Реализовать приложение (оконное или web), позволяющее:
    - Инициировать выполнение протокола на создание секретного чата с указанием используемого симметричного алгоритма шифрования;
    - Инициировать выполнение протокола на подключение к ранее созданному секретному чату;
    - Инициировать выполнение протокола на отключение от секретного чата, к которому в данный момент существует подключение;
    - Генерировать вектор инициализации (IV) для его применения в режимах шифрования: CBC, PCBC, CFB, OFB, CTR, Random Delta;
    - Организовать функционал для выбора данных, подвергаемых шифрованию и передаче второй стороне, участвующей в секретном чате:
        - ввод текста в текстовое поле;
        - выбор файла с данными при помощи стандартного диалога выбора файла.
    - Многопоточно (по возможности) шифровать данные сеансовым ключом симметричного алгоритма (с использованием одного из режимов шифрования: ECB, CBC, PCBC, CFB, OFB, CTR, Random Delta; также с использованием режима набивки: Zeros, ANSI X.923, PKCS7, ISO 10126);
    - Инициировать выполнение протокола передачи вычисленного шифротекста на сторону серверного приложения при помощи нажатия кнопки на UI;
    - Инициировать выполнение протокола получения шифротекста со стороны серверного приложения в фоновом режиме;
    - Многопоточно (по возможности) дешифровать полученный шифротекст распределённым между сторонами сеансовым ключом симметричного алгоритма (с учётом режима шифрования и режима набивки, применённых при операции шифрования);
    - Отображать список активных секретных чатов текущего пользователя;
    - Отображать переданные и полученные сообщения в надлежащем виде (текст как текст, картинки как картинки, остальные файлы - как компоненты UI, позволяющие сохранить файлы на локальное устройство при помощи стандартного диалога сохранения файла;

- Сохранять переданные и полученные сообщения в локальной СУБД, устройство и взаимодействие с которой определите самостоятельно;
- Отображать прогресс операций шифрования / дешифрования / передачи данных / получения данных при помощи элементов управления типа ProgressBar;
- Инициировать отмену операции шифрования / дешифрования / передачи данных / получения данных по запросу пользователя;
- Инициировать выполнение протокола на отключение одного или обоих клиентов от секретного чата.

Приложение должно иметь интуитивно понятный и удобный пользовательский интерфейс. Поведение клиентского приложения, приводящее к аварийной ситуации, не допускается.

2. Теоретическая часть

2.1 RC5

RC5 - это блочный шифр, разработанный Роном Ривестом из компании RSA Security, с переменным количеством раундов, длиной блока и длиной ключа.

В алгоритме используются три примитивных операции и их инверсии:

- Сложение по модулю 2^w;
- Побитовое исключающее ИЛИ (XOR);
- Операции циклического сдвига на переменное число бит (x <<< y).

2.1.1 Параметры

Так как алгоритм RC5 имеет переменные параметры, то для спецификации алгоритма с конкретными параметрами принято обозначение RC5-W/R/b:

- w - половина длины блока в битах, возможные значения 16, 32, 64;
- r - число раундов, возможные значения от 0 до 255;
- b - длина ключа в байтах, возможные значения от 0 до 255.

2.1.2 Расширение ключа

Процедура расширения ключа состоит из четырёх этапов:

1) генерация констант;
2) разбиение ключа на слова;
3) построение таблицы расширенных ключей;
4) перемешивание.

2.1.2.1 Генерация констант

Для заданного параметра W генерируются псевдослучайные величины, используя две математические константы: e (экспонента), f (Золотое сечение).

$$Q_w = Odd\left((f-1)\times 2^w\right)$$

$$P_w = Odd\left((e-1)\times 2^w\right),$$

где Odd(x) - это округление до ближайшего нечетного целого.

Для w = 16, 32, 64 получаются следующие константы:

$$P_{16} = B7E1_{16}$$

$$Q_{16} = 9E37_{16}$$

$$P_{32} = B7E15163_{16}$$

$$Q_{32} = 9E3779B9_{16}$$

$$P_{64} = B7E151628AED2A6B_{16}$$

$$Q_{64} = 9E3779B97F4A7C15_{16}$$

2.1.2.2 Разбиение ключа на слова

На данном этапе происходит разбиение массива байт ключа $K_0 \dots K_{b-1}$ на слова размером W/8 байт. Если количество байт в ключе b, не кратно W/8 байт, то результирующий массив L дополняется нулевыми байтами до ближайшего размера, кратного W/8 случае, если b = 0, то размер массива L устанавливается в 1 и $L_0 = 0$.

2.1.2.3 Построение таблицы расширенных ключей

Построение таблицы расширенных ключей $S_0 \dots S_{2\cdot(R+1)-1}$ выполняется следующим образом:

$$S_0 = P_w$$

$$S_{i+1} = S_i + Q_w$$

2.1.2.4 Перемешивание

Циклически N раз выполняются следующие действия:

$$G = S_i = (S_i + G + H) \lll 3$$

$$H = L_j = (L_j + G + H) \lll (G + H)$$

$$i = (i + 1) \bmod (2 \cdot (R + 1))$$

$$j = (j + 1) \bmod c$$

Причем G, H, i, j – временные переменные, начальные значения которых равны 0. Количество итераций цикла N – это максимальное из двух значений 3·c и (6·(R + 1)).

2.1.3 Шифрование

Исходный текст разделяется на две равные половины: левую (A) и правую (B). Перед первым раундом выполняются операции наложения расширенного ключа на шифруемые данные:

$$A = (A + S_0) \bmod 2^w$$

$$B = (B + S_1) \bmod 2^w$$

В каждом раунде выполняются следующие действия:

$$A = ((A \oplus B) \lll B) + S_{2i}$$

$$B = ((B \oplus A) \lll A) + S_{2i+1}$$

2.1.4 Расшифрование

Для дешифрования используются обратные операции. Для i = R, R−1, … , 1 выполняются следующие операции:

$$B = ((B - S_{2i+1}) \ggg A) \oplus A$$

$$A = ((A - S_{2i}) \ggg B) \oplus B$$

После выполнения всех раундов, исходное сообщение находится из выражения:

$$B = (B - S_1) \bmod 2^w$$

$$A = (A - S_0) \bmod 2^w$$

2.2 RC6

RC6 — симметричный блочный криптографический алгоритм, производный от алгоритма RC5. Был создан Роном Ривестом, Мэттом Робшау и Рэем Сиднеем для удовлетворения требований конкурса Advanced Encryption Standard (AES). Алгоритм был одним из пяти финалистов конкурса, был также представлен NESSIE и CRYPTREC. Является собственническим (проприетарным) алгоритмом, и запатентован RSA Security.

Вариант шифра RC6, заявленный на конкурс AES, поддерживает блоки длиной 128 бит и ключи длиной 128, 192 и 256 бит, но сам алгоритм, как и RC5, может быть сконфигурирован для поддержки более широкого диапазона длин как блоков, так и ключей (от 0 до 2040 бит). RC6 очень похож на RC5 по своей структуре и также довольно прост в реализации.

2.2.1 Параметры

Так же, как и RC5, RC6 — полностью параметризированная семья алгоритмов шифрования. Для спецификации алгоритма с конкретными параметрами, принято обозначение RC6-w/r/b, где

- w — длина машинного слова в битах;
- r — число раундов;
- b — длина ключа в битах. Возможные значения 0..255 бит.

2.2.2 Расширение ключа

Процедура расширения ключа аналогична RC5, состоит из четырёх этапов:

1) генерация констант;
2) разбиение ключа на слова;
3) построение таблицы расширенных ключей;
4) перемешивание.

2.2.2.1 Генерация констант

Для заданного параметра W генерируются псевдослучайные величины, используя две математические константы: e (экспонента), f (Золотое сечение).

$$Q_w = Odd\left((f-1)\times 2^w\right)$$

$$P_w = Odd\left((e-2)\times 2^w\right),$$

где Odd(x) - это округление до ближайшего нечетного целого.

Для w = 32 получаются следующие константы:

$$P_{32} = 9E3779B9_{16}$$

$$Q_{32} = B7E15163_{16}$$

2.2.2.2 Построение таблицы расширенных ключей

Следующие этапы построения таблицы расширенных ключей, такие как разбиение ключа, составление таблицы и перемешивание аналогично RC5. Отличие состоит в том, что большее количество слов получено из предоставленного пользователем ключа для использования в течение шифрования и расшифровки (2 · (R + 2)).

2.2.3 Шифрование

RC6 работает с четырьмя w-битными регистрами A, B, C и D, которые содержат входной исходный текст и выходной шифрованный текст в конце шифрования.

Перед раундами выполняются операции:

$$B = (B + S_0)$$

$$D = (D + S_1)$$

В каждом раунде выполняются следующие действия:

$$t = (B(2B + 1)) \lll \lg w$$

$$u = (D(2D + 1)) \lll \lg w$$

$$A = ((A \oplus t) \lll u) + S_{2i}$$

$$C = ((C \oplus u) \lll t) + S_{2i+1}$$

$$(A, B, C, D) = (B, C, D, A)$$

После выполнения раундов выполняются операции:

$$A = (A + S_{2r+2})$$

$$C = (C + S_{2r+3})$$

2.2.4 Расшифрование

Перед раундами выполняются операции:

$$C = (C - S_{2r+3})$$

$$A = (A - S_{2r+2})$$

В каждом раунде выполняются следующие действия:

$$(A, B, C, D) = (D, A, B, C)$$

$$u = (D(2D + 1)) \lll \lg w$$

$$t = (B(2B + 1)) \lll \lg w$$

$$C = ((C - S_{2i+1}) \gg t) \bigoplus u$$

$$A = ((A - S_{2i}) \gg u) \bigoplus t$$

После выполнения раундов выполняются операции:

$$D = (D - S_1)$$

$$B = (B - S_0)$$

3. Практическая часть

3.1 Архитектура приложения

Приложение-мессенджер представляет собой клиент-серверную архитектуру. В качестве хранилища данных используются базы данных PostgreSQL, развернутые на стороне клиента для конфиденциальности и безопасности хранения данных. В качестве брокера сообщений для безопасного общения в чатах используются брокеры Apache Kafka, администрируемые на стороне сервера. Для шифрования и дешифрования данных используется собственноручно-реализованная модуль-библиотека, содержащая все необходимые для этого инструменты и удобный кодовый интерфейс.

3.1.1 Сервер

Сервер – центральная часть комплекса приложений. Он обеспечивает:

- создание чатов;
- подключение к существующим чатам;
- удаление чатов;
- актуальное отображение списка доступных чатов клиентам
- хранение метаданных о пользователях и чатах (non-sensetive)

Так как для безопасного общения используется кластер Kafka, сервер отвечает за регистрацию новых топиков для каждого отдельного чата, а также их последующее удаление. Сервер хранит метаданные о чатах и прикрепленных к ним топиках, однако не имеет доступа к ключам шифрования, что делает общение в чате максимально приватным.

Архитектурно сервер реализован по паттерну проектирования трехслойной архитектуры. API сервера дает полный функционал для регистрации и взаимодействия с чатами, представлено в листинге 1.

Листинг 1. API сервера

```
@RestController
@RequestMapping("/chat")
@RequiredArgsConstructor
@Validated
public class ServerChatController {
```

```
private final ChatService chatService;


@PostMapping
public ResponseEntity<Object> createChat(
        @Min(0)
        @RequestHeader(value = "Client-Id")
        Long clientId,
        @NotBlank
        @RequestHeader(value = "Client-Host")
        String clientHost,
        @Min(1)
        @Max(65535)
        @RequestHeader(value = "Client-Port")
        int clientPort,


        @Valid
        @RequestBody
        CreateChatRequest createChatRequest
) {
    try {
        var chatServiceResponse = chatService.createChat(
                createChatRequest.getChatName(),
                createChatRequest.getEncryptionAlgorithm(),
                ClientInfo.builder()
                        .settings(createChatRequest.getClientSettings()
                      )
                        .id(clientId)
                        .host(clientHost)
                        .port(clientPort)
                        .publicKey(null)
                        .build());
        return ResponseEntity
                .ok(chatServiceResponse);


    } catch (IllegalArgumentException e) {
        return ResponseEntity
                .status(HttpStatus.CONFLICT)
                .body(e.getMessage());
    }
}


@PatchMapping
public ResponseEntity<Object> joinChat(
        @Min(0)
        @RequestHeader(value = "Client-Id")
        Long clientId,
        @NotBlank
        @RequestHeader(value = "Client-Host")
        String clientHost,
        @Min(1)
        @Max(65535)
        @RequestHeader(value = "Client-Port")
        int clientPort,
```

```
        @NotNull
        @RequestBody
        JoinChatRequest joinChatRequest
) {
    try {
        var chatServiceResponse = chatService.joinChat(
                joinChatRequest.getChatName(),
                ClientInfo.builder()
                        .settings(joinChatRequest.getClientSettings())
                        .id(clientId)
                        .host(clientHost)
                        .port(clientPort)
                        .publicKey(null)
                        .build());
        return ResponseEntity
                .ok(chatServiceResponse);


    } catch (InvalidKeyException e) {
        return ResponseEntity
                .status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body(e.getMessage());
    } catch (IllegalArgumentException e) {
        return ResponseEntity
                .status(HttpStatus.NOT_FOUND)
                .body(e.getMessage());
    } catch (IllegalStateException e) {
        return ResponseEntity
                .status(HttpStatus.FORBIDDEN)
                .body(e.getMessage());
    } catch (IllegalCallerException e) {
        return ResponseEntity
                .status(HttpStatus.CONFLICT)
                .body(e.getMessage());
    }
}


@PutMapping
public ResponseEntity<Object> exchangePublicKey(
        @Min(0)
        @RequestHeader(value = "Client-Id")
        Long clientId,
        @NotBlank
        @RequestHeader(value = "Chat-Name")
        String chatName,


        @Min(1)
        @RequestBody
        BigInteger publicKey
) {
    try {
        var exchangeResponse = chatService.exchangePublicKey(
                chatName,
                clientId,
```

```
                publicKey
        );
        return ResponseEntity
                .ok(exchangeResponse);
    } catch (IllegalArgumentException e) {
        return ResponseEntity
                .status(HttpStatus.NOT_FOUND)
                .body(e.getMessage());
    } catch (IllegalCallerException e) {
        return ResponseEntity
                .status(HttpStatus.FORBIDDEN)
                .body(e.getMessage());
    }
}


@DeleteMapping
public ResponseEntity<Object> exitChat(
        @Min(0)
        @RequestHeader(value = "Client-Id")
        Long clientId,
        @NotBlank
        @RequestHeader(value = "Chat-Name")
        String chatName
) {
    try {
        chatService.exitChat(chatName, clientId);
        return ResponseEntity
                .ok()
                .build();
    } catch (IllegalArgumentException e) {
        return ResponseEntity
                .status(HttpStatus.NOT_FOUND)
                .body(e.getMessage());
    } catch (IllegalCallerException e) {
        return ResponseEntity
                .status(HttpStatus.FORBIDDEN)
                .body(e.getMessage());
    }
}


@GetMapping("/list")
public ResponseEntity<List<ListChatsResponse>> listChats() {
    var chatList = chatService.listChats();
    List<ListChatsResponse> responseList = chatList.isEmpty()
            ? List.of() : chatList.stream()
            .map(chat -> ListChatsResponse.builder()
                    .chatName(chat.getKey())
                    .encryptionAlgorithm(chat.getValue())
                    .build())
            .toList();
    return ResponseEntity
            .ok(responseList);
}


@GetMapping("/is-registered/{chat-name}")
```

```
    public ResponseEntity<Boolean> isChatRegistered(
            @NotBlank
            @PathVariable("chat-name")
            String chatName
    ) {
        return ResponseEntity
                .ok(chatService.isChatRegistered(chatName));
    }
}
```

Сервер имеет возможность изменения конфигурации, как через конфигурационный файл, так и через переменные окружения. Пример конфигурационного файла представлен в листинге 2.

Листинг 2. Конфигурации сервера

```
app-configs:
  host: ${APP_HOST:localhost}
  port: ${APP_PORT:8080}
  clients:
    protocol: ${CLIENTS_PROTOCOL:HTTP}
    root-endpoint: ${CLIENTS_ROOT_ENDPOINT:/client-chat}
    public-key-endpoint: ${CLIENTS_PUBLIC_KEY_ENDPOINT:/public-key}
    notify-exit-endpoint: ${CLIENTS_NOTIFY_EXIT_ENDPOINT:/client-exit}
  kafka:
    bootstrap-servers: ${KAFKA_BOOTSTRAP_SERVERS:localhost:9093}
    topics-replication-factor: ${KAFKA_TOPICS_REPLICATION_FACTOR:2}
    topics-postfix: ${KAFKA_TOPICS_POSTFIX:-chat}
  diffie-hellman:
    bit-length: ${DIFFIE_HELLMAN_BIT_LENGTH:100}

spring:
  application:
    name: ${APP_NAME:server}
  kafka:
    bootstrap-servers: ${app-configs.kafka.bootstrap-servers}
    admin:
      client-id: ${KAFKA_ADMIN_CLIENT_ID:server-admin}

springdoc:
  swagger-ui:
    path: /endpoints

server:
  port: ${app-configs.port}
```

Полный исходный код сервера размещен в репозитории GitHub в приложении.

3.1.2 Клиент

Клиент – веб-приложение, позволяющее пользователю обмениваться данными в мессенджере. Он предоставляет удобный интерфес с со следующими возможностями:

- создавать чаты;
- подключаться к чатам;
- получать информацию об уже созданных чатах;
- конфигурировать пользовательские настройки шифрования;
- Конфигурировать настройки чата;
- Обмениваться данными в чате (текст, изображения, файлы).

Конфигурация пользовательских настроек предполагает выбор настроек шифрования, таких как режим шифрования (ECB, CBC, PCBC, CFB, OFB, CTR, Random Delta) режима набивки (Zeros, ANSI X.923, PKCS7, ISO10126), и сгенерированный вектор инициализации. Конфигурация настроек чата предполагает ввод названия чата и алгоритма шифрования с различными параметрами, общего для обоих клиентов чата.

Архитектурно клиент реализован по паттерну проектирования трехслойной архитектуры с добавлением слоя веб-интерфейса (view). Взаимодействие с локальной базой данных обеспечено на уровне репозитория. Настройки базы данных и работы с ней конфигурируемы.

Клиент имеет 3 главных интерфейса взаимодействия:

- /hello – начальный приветственный интерфейс, конфигурация имени пользователя;
- /menu – меню мессенджера, ответственное за создание и конфигурацию чатов, просмотр списка существующих чатов, подключение к существующим чатам, конфигурацию пользовательских настроек при переходе в чат;
- /chat – интерфейс самого чата, с возможностью отправки сообщений и файлов, просмотр настроек чата, своих конфигураций и конфигураций собеседника

Клиент имеет возможность изменения конфигурации, как через конфигурационный файл, так и через переменные окружения. Пример конфигурационного файла представлен в листинге 3.

Листинг 3. Конфигурации клиента

```
app-configs:
  host: ${APP_HOST:localhost}
  port: ${APP_PORT:8082}
  gateway-endpoint: ${GATEWAY_ENDPOINT:/hello}
  server:
    protocol: ${SERVER_PROTOCOL:HTTP}
    host: ${SERVER_HOST:localhost}
    port: ${SERVER_PORT:8080}
    chat:
      endpoint: ${SERVER_CHAT_ENDPOINT:/chat}
    chat-list:
      endpoint: ${SERVER_CHAT_LIST_ENDPOINT:/chat/list}
      update-interval: ${SERVER_CHAT_LIST_UPDATE_INTERVAL:3000ms}
  file-upload:
    in-memory: ${FILE_UPLOAD_IN_MEMORY:true}
    max-file-size: ${FILE_UPLOAD_MAX_FILE_SIZE:15728640}  # 15 Mb
  encryption:
    private-key-bit-length: ${ENCRYPTION_PRIVATE_KEY_BIT_LENGTH:1024}
  kafka:
    max-request-size: ${KAFKA_MAX_REQUEST_SIZE:15728641}  # 15 Mb (file-
max-size)  +  1 mb for message metadata  [producer prop]
    fetch-max-bytes: ${KAFKA_FETCH_MAX_BYTES:15728641}  # 15 Mb (file-max-
size)  +  1 mb for message metadata  [consumer prop]
  database:
    messages:
      save-encrypted: ${DB_MESSAGES_SAVE_ENCRYPTED:true}

spring:
  application:
    name: ${APP_NAME:client}
  liquibase:
    enabled: false
  servlet:
    multipart:
      max-file-size: ${app-configs.file-upload.max-file-size}
  spring:
    json:
      trusted:
        packages: "*"
  datasource:
    url:
jdbc:postgresql://${DB_HOST:localhost}:${DB_PORT:5432}/${DB_DATABASE:crypto
_messenger}
    username: ${DB_USERNAME:eflerrr}
    password: ${DB_PASSWORD:heyheyhey}
    driver-class-name: org.postgresql.Driver
  jpa:
    hibernate:
      ddl-auto: validate
    database-platform: org.hibernate.dialect.PostgreSQLDialect
    properties:
      hibernate:
        dialect: org.hibernate.dialect.PostgreSQLDialect
        format_sql: true
    show-sql: false
springdoc:
```

```
  swagger-ui:
    path: /endpoints

server:
  port: ${app-configs.port}

vaadin:
  frontend:
    hotdeploy: true
  url-mapping: /*
  excludeUrls: "/swagger-ui/**, /v3/api-docs/**, ${springdoc.swagger-
ui.path}, /client-chat/**"
```

Полный исходный код клиента размещен в репозитории GitHub в приложении.

3.1.3 Библиотека-модуль шифрования

Библиотека-модуль шифровки данных является набором инструментов для успешного шифрования и дешифрования данных на стороне клиентов. Библиотека содержит:

- низкоуровневые утилиты для работы с битами;
- сущности-шифровальщики с реализацией на каждый алгоритм шифрования;
- удобный кодовый интерфейс-менеджер для шифрования данных и файлов.

Полный исходный код шифровальщиков размещен в листингах 4 и 5 в приложении. Ниже в листинге 6 приведен исходный код менеджера шифрования.

Листинг 6. EncryptorManager

```
public class EncryptorManager {

    protected final AEncryptMode kernelMode;
    protected final IPadding padding;
    protected final int blockLength;

    public EncryptorManager(
            byte[] key,
            IEncryptor encryptor,
            EncryptionMode mode,
            PaddingType type,
            byte[] initializationVector) {
        kernelMode = getMode(mode, encryptor.setKey(key),
initializationVector);
        padding = getPadding(type);
        blockLength = encryptor.getBlockLength() / Byte.SIZE;
    }
```

```
    public EncryptorManager(
            byte[] key,
            EncryptionAlgorithm algorithm,
            EncryptionMode mode,
            PaddingType type,
            byte[] initializationVector) {
        this(key, algorithm.createEncryptorInstance(), mode, type,
initializationVector);
    }

    public byte[] encrypt(byte[] plain) {
        var withPadding = padding.makePadding(plain, blockLength);
        return kernelMode.encrypt(withPadding);
    }

    public byte[] decrypt(byte[] encoded) {
        return padding.undoPadding(kernelMode.decrypt(encoded));
    }

    public void encryptFile(String inputFile, String outputFile) throws
IOException {
        var data = Files.readAllBytes(Paths.get(inputFile));
        var encrypted = encrypt(data);
        Files.write(Paths.get(outputFile), encrypted);
    }

    public void decryptFile(String inputFile, String outputFile) throws
IOException {
        var data = Files.readAllBytes(Paths.get(inputFile));
        var decrypted = decrypt(data);
        Files.write(Paths.get(outputFile), decrypted);
    }

}
```

Полный исходный код модуля шифрования размещен в репозитории GitHub в приложении.

3.2 Описания взаимодействий

3.2.1 Синхронизация клиентов, обмен ключами

Синхронизация метаданных клиентов и обмен ключами происходит посредством нескольких REST-запросов на сервер в несколько этапов, включая протокол обмена ключами Диффи-Хеллмана, попутно обновляя актуальную информацию в UI. Подробная схема клиент-серверного взаимодействия представлена на рисунке 1.

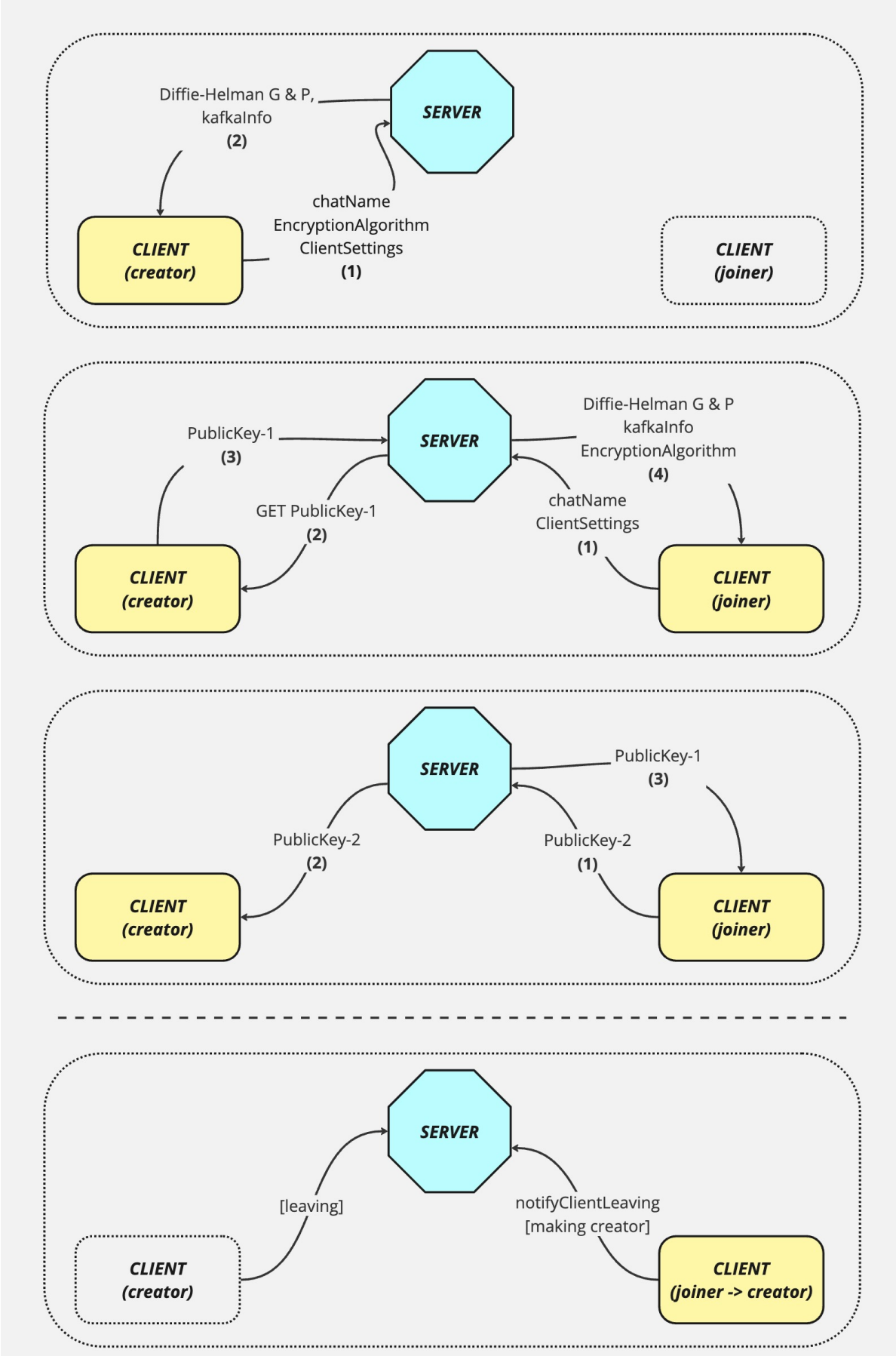

Рисунок 1. Схема клиент-серверного взаимодействия

3.2.2 Общение через брокеров Apache Kafka

После этапов синхронизации и обмена ключами клиенты знают информацию о топике чата для отправки данных. Все данные, попадающие в топик предварительно шифруются на стороне клиента. Все метаданные, поступающие в топик с зашифрованным сообщением являются открытыми и не имеют смысла, не зная содержимое сообщения. Ключи шифрования хранятся строго на стороне клиента. Вычитанные сообщения дешифруются с учетом конфигураций чата и настроек шифрования собеседника.

3.3 Используемые технологии

Используемые технологии представлены в таблице 1.

Таблица 1. Используемые технологии

Язык программирования	Java
Фреймворк	Spring (Spring Boot), Vaadin
Web-интерфейс	Vaadin
Web-сервер	Apache Tomcat
Хранилище данных	PostgreSQL
Миграции	Liquibase
Контейнерезация	Docker
ORM	Spring Data JPA (Hibernate)
Брокер сообщений	Apache Kafka
Работа с JSON	Jackson
Конфигурация приложений	Spring Config

4. Вывод

В ходе выполнения данного курсового проекта было разработано приложение для общения в секретных чатах с использованием современных и надёжных технологий. Применение языка программирования Java в сочетании с фреймворком Spring Boot позволило создать масштабируемое и удобно поддерживаемое серверное решение. Фронтенд-часть, реализованная с помощью Vaadin, обеспечивает простой и интуитивно понятный пользовательский интерфейс.

Использование Docker для развертывания служб PostgreSQL и Kafka гарантирует лёгкость в управлении зависимостями и консистентность окружения на разных стадиях разработки и развертывания. Это также способствует более эффективному масштабированию приложения и его компонентов. Благодаря интеграции с системами управления базами данных PostgreSQL через Spring Data, приложение эффективно обрабатывает информацию о клиентах и сообщениях, обеспечивая быстрый доступ и высокую производительность. Kafka используется для обмена сообщениями между пользователями в чатах, что поддерживает высокую производительность и отказоустойчивость коммуникаций.

Модульность проекта позволяет гибко расширять и масштабировать инстансы клиентов и серверов.

Таким образом, проект не только достиг поставленных целей, но и предоставил ценный опыт в области разработки современных приложений для защищённого общения.

5. Список используемых источников

1. Смарт, Н. Криптография / Н. Смарт ; пер. с англ. С. А. Кулешова ; под ред. С. К. Ландо. – Москва : Техносфера, 2005. – 528 с. – ISBN 5-94836-043-1.
2. Rivest, R. L. The RC5 Encryption Algorithm / R. L. Rivest ; MIT – USA : MIT Laboratory for Computer Science, 1997. – 12 с.

6. Приложение

Листинг 4. RC5Encryptor

```
public class RC5Encryptor implements IEncryptor {

    @Getter
    protected final int blockLength;
    protected final int keyLength;
    protected final int rounds;
    protected long[] roundKeys;
    protected int roundKeysCount;
    protected Map<Integer, Pair<Long, Long>> constants = Map.of(
            16, Pair.of(0xB7E1L, 0x9E37L),
            32, Pair.of(0xB7E15163L, 0x9E3779B9L),
            64, Pair.of(0xB7E151628AED2A6BL, 0x9E3779B97F4A7C15L)
    );


    protected long[] getWords(byte[] key) {
        int wordLength = blockLength / 2 / Byte.SIZE;
        int countWords = (keyLength + wordLength - 1) / wordLength;
        long[] result = new long[countWords];
        for (int i = 0; i < countWords; i++) {
            result[i] = Utils.getBitsFrom(
                    key, i * wordLength * Byte.SIZE, wordLength * Byte.SIZE
            );
        }
        return result;
    }

    protected long[] getSubKeys(int countSubKeys) {
        int halfBlockLength = blockLength / 2;
        long p = constants.get(halfBlockLength).getLeft();
        long q = constants.get(halfBlockLength).getRight();
        long[] result = new long[countSubKeys];
        result[0] = p;
        for (int i = 1; i < countSubKeys; i++) {
            result[i] = Utils.additionMod(result[i - 1], q,
halfBlockLength);
        }
        return result;
    }

    protected byte[][] splitTwoParts(byte[] block) {
        int halfBlockLengthInBytes = blockLength / 2 / Byte.SIZE;
        byte[][] result = new byte[2][halfBlockLengthInBytes];
        System.arraycopy(block, 0, result[0], 0, halfBlockLengthInBytes);
        System.arraycopy(block, halfBlockLengthInBytes, result[1], 0,
halfBlockLengthInBytes);
        return result;
    }

    protected byte[] clayTwoParts(long left, long right, int sizeResult) {
        byte[] leftResult = new byte[sizeResult / 2];
        byte[] rightResult = new byte[sizeResult / 2];
        for (int i = 0; i < sizeResult / 2; i++) {
```

```
            leftResult[sizeResult / 2 - i - 1] = (byte) ((left >> (i *
Byte.SIZE)) & ((1 << Byte.SIZE) - 1));
            rightResult[sizeResult / 2 - i - 1] = (byte) ((right >> (i *
Byte.SIZE)) & ((1 << Byte.SIZE) - 1));
        }
        byte[] result = new byte[sizeResult];
        System.arraycopy(leftResult, 0, result, 0, sizeResult / 2);
        System.arraycopy(rightResult, 0, result, sizeResult / 2, sizeResult
/ 2);
        return result;
    }

    protected long[] expandKey(byte[] key, int countSubKeys) {
        long[] words = getWords(key);
        long[] subKeys = getSubKeys(countSubKeys);
        int sizeHalfBlockInBits = blockLength / 2;
        int countWordsSArray = subKeys.length;
        int countWords = words.length;
        int i = 0;
        int j = 0;
        long a = 0;
        long b = 0;
        for (int counter = 0; counter < 3 * Integer.max(countWordsSArray,
countWords); counter++) {
            a = subKeys[i] = Utils.cycleLeftShift(
                    Utils.additionMod(Utils.additionMod(
                                    subKeys[i], a, sizeHalfBlockInBits),
                            b, sizeHalfBlockInBits),
                    3, sizeHalfBlockInBits);
            b = words[j] = Utils.cycleLeftShift(
                    Utils.additionMod(Utils.additionMod(
                                    words[j], a, sizeHalfBlockInBits),
                            b, sizeHalfBlockInBits),
                    Utils.additionMod(a, b, sizeHalfBlockInBits),
sizeHalfBlockInBits);
            i = (i + 1) % countWordsSArray;
            j = (j + 1) % countWords;
        }
        return subKeys;
    }

    protected boolean checkBlockLength(int blockLength) {
        return blockLength == 32 || blockLength == 64 || blockLength ==
128;
    }

    protected boolean checkKeyLength(int keyLength) {
        return keyLength > 0 && keyLength < 256;
    }

    protected boolean checkRounds(int rounds) {
        return rounds > 0 && rounds < 256;
    }


    public RC5Encryptor(int blockLength, int keyLength, int rounds) {
        if (!checkBlockLength(blockLength)) {
            throw new IllegalArgumentException("Invalid block length!");
```

```
        }
        if (!checkKeyLength(keyLength)) {
            throw new IllegalArgumentException("Invalid key length!");
        }
        if (!checkRounds(rounds)) {
            throw new IllegalArgumentException("Invalid rounds count!");
        }

        this.blockLength = blockLength;
        this.keyLength = keyLength;
        this.rounds = rounds;
        this.roundKeys = null;
        this.roundKeysCount = 2 * (rounds + 1);
    }

    public RC5Encryptor() {
        this(128, 16, 12);
    }

    @Override
    public byte[] encrypt(byte[] block) {
        if (roundKeys == null) {
            throw new NullPointerException("Round keys are not configured
before encryption!");
        }
        int halfBlockLength = blockLength / 2;
        var halfParts = splitTwoParts(block);

        long a = Utils.additionMod(Utils.bytesToLong(halfParts[0]),
roundKeys[0], halfBlockLength);
        long b = Utils.additionMod(Utils.bytesToLong(halfParts[1]),
roundKeys[1], halfBlockLength);

        for (int i = 1; i < rounds; i++) {
            a = Utils.additionMod(
                    Utils.cycleLeftShift((a ^ b), b, halfBlockLength),
                    roundKeys[2 * i], halfBlockLength);
            b = Utils.additionMod(
                    Utils.cycleLeftShift((a ^ b), a, halfBlockLength),
                    roundKeys[2 * i + 1], halfBlockLength);
        }
        return clayTwoParts(a, b, blockLength / Byte.SIZE);
    }

    @Override
    public byte[] decrypt(byte[] block) {
        if (roundKeys == null) {
            throw new NullPointerException("Round keys are not configured
before decryption!");
        }
        int halfBlockLength = blockLength / 2;
        var halfParts = splitTwoParts(block);

        long a = Utils.bytesToLong(halfParts[0]);
        long b = Utils.bytesToLong(halfParts[1]);

        for (int i = rounds - 1; i >= 1; i--) {
            b = Utils.cycleRightShift(
```

```
                    Utils.subtractionMod(b, roundKeys[2 * i + 1],
halfBlockLength),
                    a, halfBlockLength) ^ a;
            a = Utils.cycleRightShift(
                    Utils.subtractionMod(a, roundKeys[2 * i],
halfBlockLength),
                    b, halfBlockLength) ^ b;
        }

        b = Utils.subtractionMod(b, roundKeys[1], halfBlockLength);
        a = Utils.subtractionMod(a, roundKeys[0], halfBlockLength);

        return clayTwoParts(a, b, blockLength / Byte.SIZE);
    }

    @Override
    public IEncryptor setKey(byte[] key) {
        if (key == null) {
            throw new NullPointerException("Key cannot be null!");
        }
        if (key.length != keyLength) {
            throw new IllegalArgumentException(
                    String.format("Invalid key length (%d != %d)!",
key.length, keyLength)
            );
        }
        this.roundKeys = expandKey(key, roundKeysCount);
        return this;
    }

}
```

Листинг 5. RC6Encryptor

```
public class RC6Encryptor extends RC5Encryptor {

    protected double log2(double digit) {
        return Math.log(digit) / Math.log(2);
    }

    protected byte[][] splitFourParts(byte[] block) {
        int quarterBlockLengthInBytes = blockLength / 4 / Byte.SIZE;
        byte[][] result = new byte[4][quarterBlockLengthInBytes];
        for (int i = 0; i < 4; i++) {
            System.arraycopy(block, quarterBlockLengthInBytes * i,
                    result[i], 0, quarterBlockLengthInBytes);
        }
        return result;
    }

    protected byte[] clayFourParts(long a, long b, long c, long d, int
sizeResult) {
        var firstPart = clayTwoParts(a, b, sizeResult / 2);
        var secondPart = clayTwoParts(c, d, sizeResult / 2);
        byte[] result = new byte[sizeResult];
        System.arraycopy(firstPart, 0, result, 0, sizeResult / 2);
```

```
        System.arraycopy(secondPart, 0, result, sizeResult / 2, sizeResult
/ 2);
        return result;
    }

    @Override
    protected boolean checkBlockLength(int blockLength) {
        return blockLength == 64 || blockLength == 128 || blockLength ==
256;
    }


    public RC6Encryptor(int blockLength, int keyLength, int rounds) {
        super(blockLength, keyLength, rounds);
        this.roundKeysCount = 2 * (rounds + 2);
        this.constants = Map.of(
                32, Pair.of(0xB7E1L, 0x9E37L),
                64, Pair.of(0xB7E15163L, 0x9E3779B9L),
                128, Pair.of(0xB7E151628AED2A6BL, 0x9E3779B97F4A7C15L)
        );
    }

    public RC6Encryptor() {
        this(256, 16, 20);
    }

    @Override
    public byte[] encrypt(byte[] block) {
        if (roundKeys == null) {
            throw new NullPointerException("Round keys are not configured
before encryption!");
        }
        int quarterBlockLength = blockLength / 4;

        var parts = splitFourParts(block);
        long a = bytesToLong(parts[0]);
        long b = additionMod(bytesToLong(parts[1]), roundKeys[0],
quarterBlockLength);
        long c = bytesToLong(parts[2]);
        long d = additionMod(bytesToLong(parts[3]), roundKeys[1],
quarterBlockLength);

        for (int i = 1; i <= rounds; i++) {
            long t = cycleLeftShift(
                    multiplicationMod(b,
                            additionMod(
                                    additionMod(b, b, quarterBlockLength),
                                    1L, quarterBlockLength),
                            quarterBlockLength),
                    (int) log2(quarterBlockLength), quarterBlockLength
            );
            long u = cycleLeftShift(
                    multiplicationMod(d,
                            additionMod(
                                    additionMod(d, d, quarterBlockLength),
                                    1L, quarterBlockLength),
                            quarterBlockLength),
                    (int) log2(quarterBlockLength), quarterBlockLength
```

```
            );
            a = additionMod(
                    cycleLeftShift(a ^ t, u, quarterBlockLength),
                    roundKeys[2 * i], quarterBlockLength);
            c = additionMod(
                    cycleLeftShift(c ^ u, t, quarterBlockLength),
                    roundKeys[2 * i + 1], quarterBlockLength);

            long tmp = a;
            a = b;
            b = c;
            c = d;
            d = tmp;
        }

        a = additionMod(a, roundKeys[2 * rounds + 2], quarterBlockLength);
        c = additionMod(c, roundKeys[2 * rounds + 3], quarterBlockLength);

        return clayFourParts(a, b, c, d, blockLength / Byte.SIZE);
    }

    @Override
    public byte[] decrypt(byte[] block) {
        if (roundKeys == null) {
            throw new NullPointerException("Round keys are not configured
before decryption!");
        }
        int quarterBlockLength = blockLength / 4;

        var parts = splitFourParts(block);
        long a = subtractionMod(bytesToLong(parts[0]), roundKeys[2 * rounds
+ 2], quarterBlockLength);
        long b = bytesToLong(parts[1]);
        long c = subtractionMod(bytesToLong(parts[2]), roundKeys[2 * rounds
+ 3], quarterBlockLength);
        long d = bytesToLong(parts[3]);

        for (int i = rounds; i >= 1; i--) {
            long tmp = d;
            d = c;
            c = b;
            b = a;
            a = tmp;

            long u = cycleLeftShift(
                    multiplicationMod(d,
                            additionMod(
                                    additionMod(d, d, quarterBlockLength),
                                    1L, quarterBlockLength),
                            quarterBlockLength),
                    (int) log2(quarterBlockLength), quarterBlockLength
            );
            long t = cycleLeftShift(
                    multiplicationMod(b,
                            additionMod(
                                    additionMod(b, b, quarterBlockLength),
                                    1L, quarterBlockLength),
                            quarterBlockLength),
```

```
                    (int) log2(quarterBlockLength), quarterBlockLength
            );

            c = cycleRightShift(
                    subtractionMod(c, roundKeys[2 * i + 1],
quarterBlockLength),
                    t, quarterBlockLength
            ) ^ u;
            a = cycleRightShift(
                    subtractionMod(a, roundKeys[2 * i],
quarterBlockLength),
                    u, quarterBlockLength
            ) ^ t;
        }

        d = subtractionMod(d, roundKeys[1], quarterBlockLength);
        b = subtractionMod(b, roundKeys[0], quarterBlockLength);

        return clayFourParts(a, b, c, d, blockLength / Byte.SIZE);
    }
}
```

Репозиторий GitHub с исходным кодом: https://github.com/Efler/crypto-messenger