DOI 10.37882/2223-2966.2025.06-2.20

СИСТЕМА БЕЗОПАСНОГО ХРАНЕНИЯ И ПЕРЕДАЧИ ФАЙЛОВ НА ОСНОВЕ РИДА—СОЛОМОНА

SECURE FILE STORAGE AND TRANSMISSION SYSTEM BASED ON REED-SOLOMON

G. Lomonosov
A. Brysin
Yu. Kirilina
V. Lavrushin

Summary. Within this work, the possibility of implementing a secure file storage system in an enterprise's local network without using centralized storage is considered to minimize the risks associated with centralized storage of files and other sensitive information. This article presents the development of an innovative system for secure storage and transmission of files based on the use of Reed—Solomon codes. The relevance of the study is determined by the increasing demands for protecting digital information, the growing volume of transmitted data, and the need to ensure its integrity when transmitting through unreliable communication channels. The main goal of the work is to create a reliable mechanism for error detection and correction that can maintain high efficiency even under significant interference. The research methodology is based on the theoretical principles of coding and Reed-Solomon algorithms, which are widely used to eliminate errors during the transmission of digital data. The article describes in detail the stages of system development, including an analysis of existing solutions, the selection of optimal coding parameters, the design of the algorithm for processing and forming redundant data, as well as the modeling of failure scenarios. The experimental part of the research was conducted using simulated test sets representing various levels of file damage, which allowed for the assessment of the error-correcting capability of the proposed method. Experimental results demonstrated that the implementation of Reed—Solomon algorithms significantly increases the reliability of the information storage and transmission system. The proposed solution guarantees the recovery of lost segments of data even at a high error level, showing performance comparable to modern counterparts in the field of information protection. Additional analysis confirmed the system's resilience to various types of impacts, making it promising for use in mission-critical applications. The discussion of the obtained results allows one to assert that the proposed system represents an effective means of ensuring the security of digital data. In conclusion, directions for further research are considered, including the expansion of functionality, integration with cryptographic methods, and adaptation to different industry requirements. The combination of theoretical foundations and practical solutions presented in the article opens new prospects for the development of comprehensive information protection systems in the context of modern cyber threats.

Keywords: centralization, decentralization, recovery, redundancy, secret sharing, encryption, broadcasting, unidirectional data transfer.

Ломоносов Георгий Алексеевич
Лаборант,
Российский технологический университет МИРЭА
personal@glomonosov.ru
Брысин Андрей Николаевич
Кандидат технических наук, доцент,
Российский технологический университет МИРЭА
brysin@rambler.ru
Кириллина Юлия Владимировна
Кандидат экономических наук, доцент,
Российский технологический университет МИРЭА
kirillina@mirea.ru
Лаврушин Вадим Максимович
Российский технологический университет МИРЭА
76187@bk.ru

Аннотация. В рамках работы рассматривается возможность реализации системы безопасного хранения файлов в локальной сети предприятия без использования централизованного хранилища для минимизации рисков, связанных с централизованным хранением файлов и другой чувствительной информации. В данной статье представлена разработка инновационной системы безопасного хранения и передачи файлов, основанной на применении кодов Рида-Соломона. Актуальность исследования определяется возрастающими требованиями к защите цифровой информации, увеличением объема передаваемых данных и необходимостью обеспечения их целостности при передаче через ненадежные каналы связи. Основной целью работы является создание надежного механизма обнаружения и коррекции ошибок, способного сохранять высокую эффективность даже в условиях значительных помех. Методология исследования базируется на теоретических принципах кодирования и алгоритмах Рида—Соломона, широко применяемых для устранения ошибок при передаче цифровых данных. В статье подробно описаны этапы разработки системы, включающие анализ существующих решений, подбор оптимальных параметров кодирования, проектирование алгоритма обработки и формирования избыточных данных, а также моделирование сценариев отказов. Экспериментальная часть исследования проводилась с использованием симулированных тестовых наборов, представляющих различные уровни повреждений файлов, что позволило оценить корректирующую способность предлагаемого метода. Результаты экспериментов показали, что внедрение алгоритмов Рида—Соломона существенно повышает надежность системы хранения и передачи информации. Предложенное решение гарантирует восстановление утраченных участков данных даже при значительном уровне ошибок, демонстрируя показатели, сопоставимые с современными аналогами в области защиты информации. Дополнительный анализ подтвердил устойчивость системы к различным видам воздействия, что делает ее перспективной для применения в критически важных приложениях. Обсуждение полученных результатов позволяет утверждать, что предложенная система представляет собой эффективное средство обеспечения безопасности цифровых данных. В заключение рассматриваются направления дальнейших исследований, включая расширение функционала, интеграцию с криптографическими методами и адаптацию к различным требованиям отрасли. Сочетание теоретических основ и практических решений, изложенных в статье, открывает новые перспективы для разработки комплексных систем защиты информации в условиях современной киберугрозы.

Ключевые слова: централизация, децентрализация, восстановление, избыточность, разделение секрета, шифрование, широковещание, однонаправленная передача данных.

На текущий момент обеспечение информационной безопасности на различных предприятиях в подавляющем большинстве случаев выстроено по централизованной модели. Централизация безусловно имеет много позитивных аспектов. В первую очередь она позволяет гораздо удобнее и эффективнее осуществлять мониторинг и администрирование. Если говорить о файловых хранилищах, то с точки зрения как пользователя, так и администратора, они являются крайне удобным и эффективным средством. Пользователи знают, в каком конкретном месте они могут получить нужные им данные, а администраторам гораздо проще обеспечивать работоспособность всего лишь одного сервера вместо множества таковых и так же удобно осуществлять разграничение доступа к разным данным разным сотрудникам и отделам. Децентрализация передачи данных особенно актуальна при обмене информации от менее защищенного сегмента сети к более защищенному в режимах однонаправленной много поточной передачи данных.

Однако вместе с очевидными удобствами такой уклон в сторону централизации ведет и к негативным аспектам. Основным недостатком централизованного хранения чувствительных данных стоит назвать наличие единой точки отказа в широком смысле слова. Кроме того, что наличие единого узла хранения данных при выходе его из строя отрежет всем сотрудникам доступ к большому числу данных на неопределенный срок, под единой точкой отказа понимается также возможность одновременной компрометации огромного объема чувствительных данных компании. Причем случиться такая компрометация может по совершенно разным причинам, а ее источником может быть как внешний, так и внутренний нарушитель. Именно возникающие вследствие централизованного хранения файлов риски и ведут к постановке различных задач по внедрению децентрализованных инструментов.

В качестве входных данных использовались сведения о кодах Рида-Соломона.

Коды Рида — Соломона — это линейные недвоичные систематические циклические коды, используемые для обнаружения и исправления ошибок в цифровых данных.

Основные параметры кодов Рида — Соломона [1, с. 684; 2, с. 745]:

- Длина кодового слова (n): общее количество символов в кодовом слове;
- Число информационных символов (k): количество символов, содержащих исходную информацию;
- Число проверочных символов ($n - k$): количество символов, добавленных для обнаружения и исправления ошибок.

Коды Рида — Соломона определяются над конечным полем $GF(q)$, где $q = 2^r$, а r — положительное целое число. Каждый символ кодового слова представляет собой элемент этого поля и может быть представлен r-битовой последовательностью.

Порождающий многочлен кода Рида — Соломона строится на основе примитивного элемента α конечного поля $GF(q)$ и имеет вид:

$$g(x) = (x - \alpha^{l_0}) * (x - \alpha^{l_0+1}) * \ldots * (x - \alpha^{l_0+d-2}),$$

где l_0 — некоторое целое число, часто принимаемое равным 1, а $d = n - k + 1$ — минимальное кодовое расстояние. Степень порождающего многочлена $(n - k)$.

Процесс кодирования заключается в умножении информационного многочлена $m(x)$ степени не более $(k - 1)$ на порождающий многочлен.

$$c(x) = m(x) * g(x),$$

где $c(x)$ — полученный кодовый многочлен степени $(n - 1)$[2, с. 745].

Задача была разделена на две части — отправка файла на хранение и получение файла из хранилища.

Перед отправкой файл разбивается на блоки одинакового размера. В случае, если последний блок данных будет меньше остальных, его конец будет заполнен нулями. Размер блока вычисляется по формулам [1, с. 684]

1. $bs = bs_{min} * \left((fs - bs_{min})^{coef} \right)$
2. $bs = max\left(bs_{min}, (bs, bs_{max}) \right)$
3. $bs = \left(\frac{(bs + a - 1)}{a} \right) * a,$

где bs — размер одного блока, bs_{min} — минимальный размер блока, bs_{max} — максимальный размер блока, $coef$ — коэффициент роста, fs — размер файла, a — значение выравнивания, выставляется в соответствии с шириной машинного слова на конкретной системе.

В результате разбиения файла формировались два массива данных — первый хранил содержимое файла, разбитое на блоки, второй — блоки восстановления, из которых в случае отсутствия блоков данных можно восстанавливать информационную часть данных.

После разбиения файла на блоки происходит процедура шифрования, применяемая к каждому блоку данных и восстановления. Шифрование осуществлялось блочным шифром «Кузнечик» [3]. Перед началом процесса шифрования формируется ключ шифрования,

который в свою очередь формируется из токена пользователя операционной системы. При первом запуске программы ею генерируются три объекта — токен пользователя, гамма и соль. Эти объекты в дальнейшем используются для шифрования и дешифрования данных, которые отправляет и принимает автоматизированная система. Все три объекта (рисунок 1). закреплены за конкретным пользователем операционной системы. По окончании процесса формирования этих сущностей, если программа запускается впервые, или их чтения формируется ключ шифрования используемого блочного шифра, после чего при помощи этого ключа шифруются все блоки данных и блоки восстановления

Рис. 1. Блок-схема процесса шифрования блоков данных

После того, как все блоки данных были зашифрованы, выполняется процедура получения хэш-суммы для каждого блока. Получение хэш-сумм выполняется при помощи функции хэширования «Стрибог» [3]. По результатам вычисления хэш-функций формируется два массива — с хэш-суммами блоков данных и блоков восстановления соответственно.

Завершающий этап отправки файла на хранение заключается в отправке данных случайно выбранному компьютеру в локальной сети, на котором запущен сервер приема данных. Сначала формируется сообщение с запросом на хранение блока данных, которое содержит хэш-сумму отправляемого блока. Это сообщение отправляется по широковещательному адресу сети на порт 62092/UDP. После отправки клиент ждет подтверждения в течение 5 секунд. При поступлении такого сообщения на сервер серверная часть программы проверяет возможность сохранить блок данных в своем хранилище. Если проверка дает положительный результат, то сервер формирует подтверждение отправки с той же хэш-суммой, после чего отправляет обратно клиенту. Клиент, получая первое подтверждение, формирует сообщение, содержащее блок и его хэш-сумму, после чего отправляет тому серверу, от которого пришло подтверждение (рисунок 2). Сервер в свою очередь принимает это сообщение и сохраняет блок в своем хранилище (рисунок 3). Эта процедура повторяется для каждого блока данных.

По завершении отправки исходный файл перезаписывается структурой, хранящей все хэш-суммы блоков и размер одного блока, сериализованной по стандарту JSON и затем закодированной при помощи алгоритма Base64. На этом этапе завершается отправка файла на хранение.

Получение файла из хранилища начинается с чтения содержимого файла с последующими декодировани-

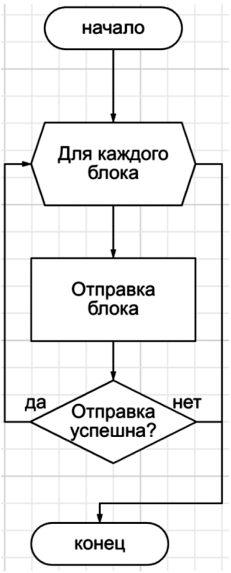

Рис. 2. Блок-схема процесса отправки данных клиентом

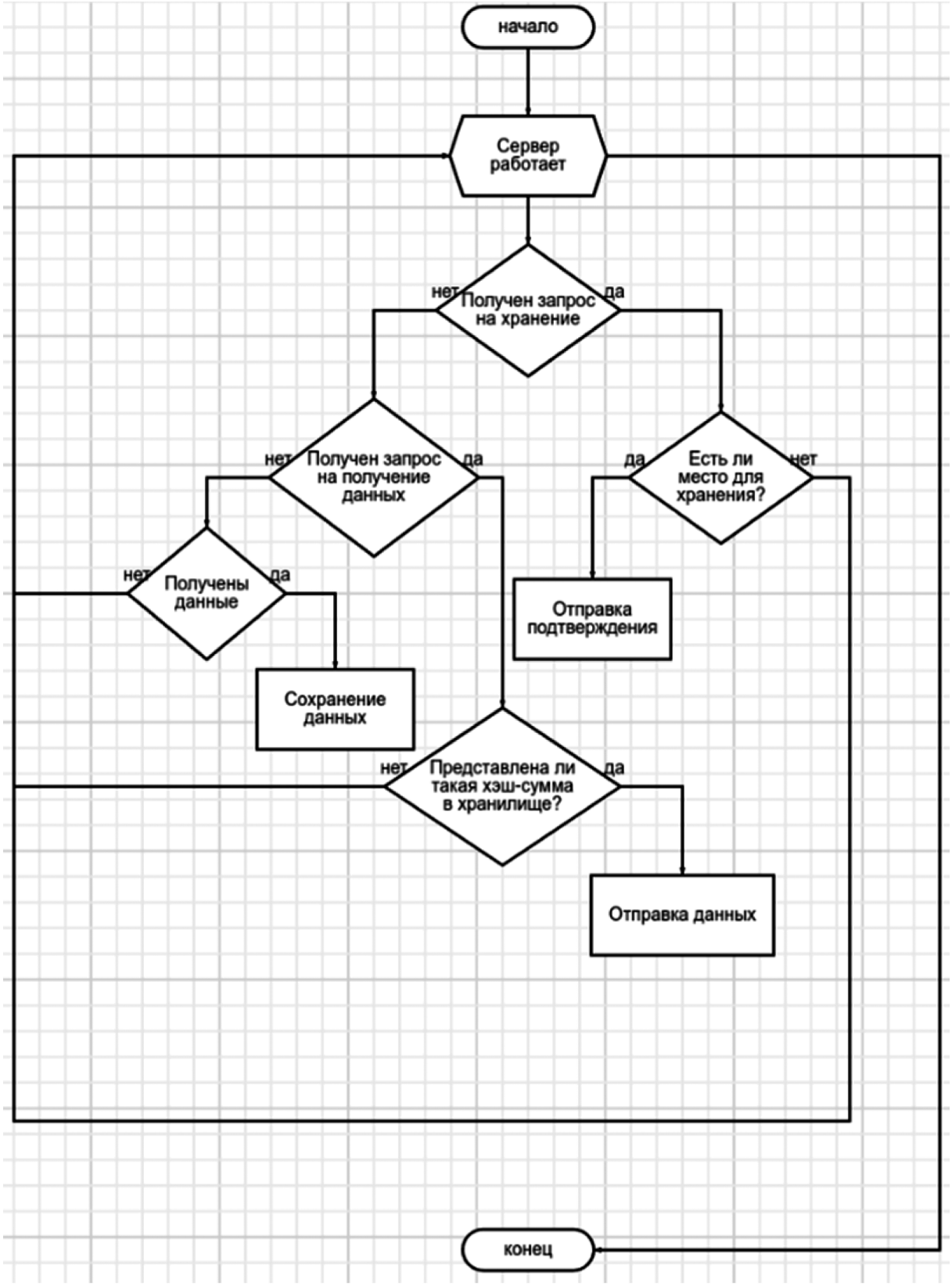

Рис. 3. Блок-схема алгоритма работы сервера

Рис. 4. Блок-схема процесса получения блоков клиентом

ем при помощи алгоритма Base64 и десериализацией по стандарту JSON.

После получения структуры, хранящей хэш-суммы блоков и размер одного блока начинается процесс получения данных из широковещательного домена. Для этого для каждой хэш-суммы формируется запрос на получение данных, содержащий хэш-сумму запрашиваемого блока. Этот запрос отправляется на широковещательный адрес сети на порт 62092/UDP [5, с. 955]. Клиент ожидает ответа в течение 5 секунд (рисунок 4). Сервер, получив такой запрос, ищет в своем хранилище данные с указанной хэш-суммой, и в случае успеха формирует сообщение (рисунок 3), содержащее хэш-сумму блока и сам блок, после чего отправляет его клиенту Важно отметить, что в случае, если клиенту не удалось по каким-либо причинам получить блок данных, то программа не завершается. Если по результатам получения блоков данных какие-то блоки не были получены, то будет предпринята попытка получить соответствующие блоки восстановления с уведомлением пользователя об этом по алгоритму, описанному выше.

Рис. 5. Блок-схема процесса дешифрования блоков

После того, как все блоки были расшифрованы, начинается процесс восстановления файла из блоков по схеме Рида — Соломона. В ходе восстановления программа воссоздает информационную часть файла, после чего проверяет, были ли дописаны нули в конец потока данных. Если были, то они удаляются. В результате этого преобразования будет получено изначальное содержимое файла.

Завершающим этапом является запись в исходный файл полученных данных. На этом процесс получения файла из хранилища завершен.

Таблица 1.

Результаты тестирования разбиения данных при передаче

Сценарий тестирования	Результаты отправки файла	Результаты получения файла	Изменения исходного файла	Примечания
Идеальная передача	Все блоки данных и восстановления успешно отправлены обоим серверам в случайном порядке	Файл был успешно получен	Не обнаружены	Блоки восстановления остались на хранении
В процессе передачи файла не был отправлен блок данных	Программа уведомила пользователя о невозможности отправки всех данных в сеть, после чего завершилась	Файл не был получен	Не обнаружены	Часть блоков может быть отправлена на хранение
В процессе передачи файла не был отправлен блок восстановления	Программа уведомила пользователя о невозможности отправки всех данных в сеть, после чего завершилась	Файл не был получен	Не обнаружены	Часть блоков может быть отправлена на хранение
Сценарий тестирования	Результаты отправки файла	Результаты получения файла	Изменения исходного файла	Примечания
В процессе получения файла не был получен блок данных	Все блоки данных и восстановления успешно отправлены обоим серверам в случайном порядке	Файл был успешно получен	Не обнаружены	Часть блоков восстановления осталась на хранении
В процессе получения файла не был блок данных и соответствующий ему блок восстановления	Все блоки данных и восстановления успешно отправлены обоим серверам в случайном порядке	Файл не был получен	Полностью перезаписан	Часть блоков данных и восстановления осталась на хранении

После того, как все блоки были получены, начинается процесс дешифрования. Процесс получения ключа аналогичен тому, который был описан для процесса шифрования (рисунок 5).

Результаты тестирования

Тестирование проводилось на сжатом при помощи алгоритма gzip tar-архиве размером 929 Кб. Тестовый стенд представлял собой локальную сеть из трех виртуальных машин:

1. Debian 12 будет выступать в роли клиента и отправлять, и запрашивать файл;
2. Ubuntu 24.04 выступает в роли сервера;
3. Fedora 41 выступает в роли сервера.

В процессе тестирования проверялись: Работоспособность в идеальных условиях, когда никакие данные не были потеряны ни при передаче на хранение, ни при получении из хранилища.

- Работоспособность в условиях, когда при передаче файла был потерян один информационный блок;
- Работоспособность в условиях, когда при получении файла не был получен блок данных;
- Работоспособность в условиях, когда при получении файла не были получены блок данных и соответствующий ему блок восстановления.

По окончании тестирования была составлена таблица результатов, представленная в Таблице 1.

Под идеальной передачей в рамках тестирования понимается ситуация, при которой все сформированные блоки были успешно отправлены, а при восстановлении файла были успешно получены все блоки данных.

Таким образом, автоматизированная система требует отправки всех сформированных блоков для корректного завершения работы и отправки файла на хранение. В процессе восстановления файла программа допускает возможность потери вплоть до всех сформированных блоков данных и восстановление всех данных из блоков восстановления, однако если вместе с блоком данных будет утерян соответствующий ему блок восстановления, то файл будет невозможно восстановить. Разработанное решение позволяет избежать возможности перехвата чувствительных данных во время их передачи, а использование встроенных систем аутентификации используемых операционных систем позволит избежать необходимости обеспечения сохранности хранимого пароля, делегировав это механизмам безопасности ОС

Реализация шифрования и использование криптографической хэш-функции позволяет свести к минимуму возможность возникновения коллизий и поиска прообраза данных, а также сразу обнаружить нарушение целостности хранимых данных. Предлагаемое решение может быть использовано при реализации однонаправленных шлюзов с разбиением потоков данных. Применение таких решений необходимо при реализации однонаправленной передачи пакетов объемом более 200 мегабайт.

ЛИТЕРАТУРА

1. Рацеев С.М. Элементы высшей алгебры и теории кодирования: учеб. пособие для вузов. СПб.: Лань, 2023. 684 с.
2. Ф.Дж. Мак-Вильямс, Н.Дж. А. Слоэн, Ф.Дж. Мак-Вильямс, Н.Дж.А. Слоэн Теория кодов, исправляющих ошибки / Ф.Дж. Мак-Вильямс, Н.Дж. А. Слоэн Ф.Дж. Мак-Вильямс, Н.Дж. А. Слоэн. — Москва: Связь, 1979. — 745 с.
3. Национальный стандарт РФ ГОСТ Р 34.11–2018 «Информационная технология. Криптографическая защита информации. Функция хэширования» (утв. приказом Федерального агентства по техническому регулированию и метрологии от 7 августа 2012 г. N 216-ст).: дата введения: 2019-06-01. — URL: https://protect.gost.ru/v.aspx?control=8&baseC=-1&page=0&month=-1&year=-1&search=&RegNum=1&DocOnPageCount=15&id=224241(дата обращения: 24.02.2025). — Текст: электронный.
4. Межгосударственный стандарт ГОСТ Р 34.12–2018 «Информационная технология. Криптографическая защита информации. Блочные шифры» (утв. приказом Федерального агентства по техническому регулированию и метрологии от 4 декабря 2018 г. N 1061-ст).: дата введения: 2019-07-01. — URL: https://protect.gost.ru/v.aspx?control=8&baseC=-1&page=0&month=-1&year=-1&search=&RegNum=1&DocOnPageCount=15&id=224244(дата обращения: 24.02.2025). — Текст: электронный
5. Таненбаум, Уэзеролл. Э.Д. Компьютерные сети / Э.Д. Таненбаум, Уэзеролл. — Санкт-Петербург: Питер, 2020. — 955 с.
6. Блэнди Дж. Программирование на языке Rust / пер. с анг. А.А. Слинкина. / Дж. Блэнди, Дж. Орендорф. — М.: ДМК Пресс, 2018. — 550 с.: ил. с. — ISBN 978-5-97060-236-2.