ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И ТЕЛЕКОММУНИКАЦИИ
INFORMATION TECHNOLOGY AND TELECOMMUNICATIONS

УДК 004.056.53

DOI: 10.21822/2073-6185-2025-52-1-134-146 Оригинальная статья /Original article

Механизм восстановления данных в результате их повреждения, заражения и/или несанкционированного изменения

Л.В. Черкесова, В.А. Савельев, Е.А. Ревякина, А.Р. Полулях, М.А. Семенцов
Донской государственный технический университет,
344000, г. Ростов-на-Дону, пл. Гагарина, 1, Россия

Резюме. Цель. Целью исследования является программный анализ методов хэширования, сжатия и восстановления информации и разработка на этой основе модуля программного комплекса. **Метод.** В ходе исследования применены алгоритмы хэширования. **Результат.** Представлен возможный функционал программного средства и организованные механизмы проверки целостности путём использования хэш-таблиц и восстановления файла из резервной копии. Модуль программного комплекса использует разработанный алгоритм, позволяющий устранить уязвимости, связанные с целостностью программы, а также значительно снизить влияние вредоносных алгоритмов на целостность файлов. Для разработанного программного средства проводится сравнительный анализ с имеющимися аналогами, а также графическое представление работоспособности алгоритма, показывающее зависимость времени от количества файлов. Для разработки программного модуля был выбран реверсивный инкрементальный алгоритм резервного копирования, как наиболее подходящий для разработанного алгоритма и более удобный в использовании. **Вывод.** Предложенный механизм восстановления данных является современным решением, обеспечивающим сохранность личных файлов в случае их повреждения. Для будущего улучшения программного средства определены основные задачи: расширение функционала программного средства; оптимизация программного кода для достижения большего быстродействия; обновление и улучшение модулей программного средства; добавление функций копирования образа диска.

Ключевые слова: хэширование, восстановление, механизм, защита, хэш-функции, SHA, RLE

Для цитирования: Л.В. Черкесова, В.А. Савельев, Е.А. Ревякина, А.Р. Полулях, М.А. Семенцов. Механизм восстановления данных в результате их повреждения, заражения и/или несанкционированного изменения. Вестник Дагестанского государственного технического университета. Технические науки. 2025; 52(1):134-146. DOI:10.21822/2073-6185-2025-52-1-134-146

Mechanism for data recovery as a result of data corruption, infection and/or unauthorized modification

L.V. Cherkesova, V.A. Savelyev, E.A. Revyakina, A.R. Polulyakh, M.A. Sementsov
Don State Technical University,
1 Gagarin Square, Rostov-on-Don 344000, Russia

Abstract. Objective. The objective of the study is software analysis of hashing, compression and recovery methods and development of a software module on this basis. **Method.** Hashing algorithms were used in the study. **Result.** The possible functionality of the software tool and organized integrity checking mechanisms by using hash tables and restoring a file from a backup copy are presented. The software module uses the developed algorithm that allows eliminating vulnerabilities associated with program integrity, as well as significantly reducing the impact of malicious algorithms on file integrity. A comparative analysis with existing analogs is carried out for the developed

software tool, as well as a graphical representation of the algorithm's performance, showing the dependence of time on the number of files. To develop the software module, a reversible incremental backup algorithm was chosen as the most suitable for the developed algorithm and more convenient to use. **Conclusion.** The proposed data recovery mechanism is a modern solution that ensures the safety of personal files in case of their damage. The main tasks for future improvement of the software tool have been defined: expanding the functionality of the software tool; optimizing the program code to achieve greater performance; updating and improving the software modules; adding disk image copy functions.

Keywords: hashing, recovery, mechanism, protection, hash-function, SHA, RLE

For citation: L.V. Cherkesova, V.A. Savelyev, E.A. Revyakina, A.R. Polulyakh, M.A. Sementsov. Mechanism for data recovery as a result of data corruption, infection and/or unauthorized modification. Herald of Daghestan State Technical University. Technical Sciences. 2025; 52(1):134-146. (In Russ) DOI:10.21822/2073-6185-2025-52-1-134-146.

Введение. В настоящее время существует огромное количество вредоносных программ, нацеленных на повреждение файлов, хранящихся на серверах и компьютерах. Проблема повреждения целостности файлов является одной из главных проблем неработающих программ и нечитаемых файлов, которые были повреждены в результате какой-либо атаки или их целостность нарушена по каким-либо другим причинам**.**

Стоит отметить, что проблема повреждения целостности, безусловно, рассматривается ведущими компаниями, но нет полной гарантии, что у файлов в системе, на сервере или в личной папке не будет нарушена целостность. Для таких целей используется резервное копирование, называемое сжатием и восстановлении информации. С момента появления первых методов хэширования и прошло довольно большое количество времени, но это не значит, что они не улучшаются – каждый последующий алгоритм, как правило, ускоряет время, необходимое для составления хэш-функции файла и усложняет структуру алгоритма во избежание повторения составленной хэш-функции.

В большинстве случаев, хэш-функция используется для создания сравнения контрольных сумм, цифровой подписи или создании уникальных идентификаторов. Крайне важно отметить, что алгоритмы сжатия и восстановления данных необходимы для создания резервных копий файлов, которые позволяет сохранить данные в случае их повреждения, будь то повреждения системы, несанкционированный доступ или намеренное воздействие вредоносных программ.

Постановка задачи. Целью исследования является программный анализ методов хэширования, сжатия и восстановления информации и разработка на этой основе модуля программного комплекса.

Методы исследования. Хэширование – это преобразование некоторого массива данных динамического размера в строку фиксированного размера, для этого используются алгоритмы хэширования или так называемые хэш-функции.

Хэширование можно применять в различных ситуациях. Алгоритм хэширования – это некоторая функция, которая на вход получает битовую строку, и на выходе выдаёт тоже битовую строку, хэш-код, но значительно меньшую по своим размерам. Однако, данный процесс необратимый, что означает – восстановление информации из хэш-кода не представляется возможным. Несмотря на такой большой минус, хэш-функции имеют значительный ряд применений, активно используемый в нынешнее время: создание уникальных идентификаторов; поиск дубликатов в наборах данных; вычисление контрольных сумм; хранение паролей; и т.д.;

В данном программном модуле, входящем в состав программного комплекса «BadCoreGuard», основное применение хэш-функции заключается в использовании контрольной суммы для содержимого файла. Стоит отметить, что крайне важным свойством

хэш-кода, что даже при незначительном изменение входной строки происходят значительные изменения выходной, из-за чего полученная хэш-функция будет отличаться от первоначальной.

Однако у хэш-кодов присутствует изъян – коллизии, ситуация, возникающая, когда при разных входных строках на выходе получается одна и та же выходная. Появление коллизии является лишь вопросом времени и зависит от используемого алгоритма хэш-функции, ведь чем больше бит используется и чем сложнее хэш-функция, тем меньше вероятность коллизии, не говоря уже о возможности решения коллизий с помощью специально созданных для этого различных методов.

На рис. 1 представлен один из вариантов решения коллизий с помощью хэш-таблиц в PHP5. Для данного программного модуля коллизии являются незначительными, так как не влияют на работоспособность самой программы.

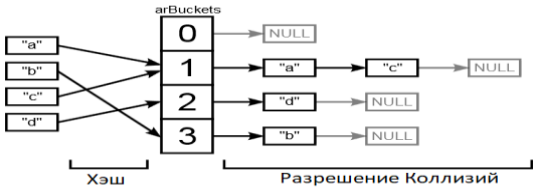

Рис. 1 – Решение коллизий с помощью хэш-таблицы
Fig. 1 – Collision resolution using a hash table

Для программного модуля было необходимо рассмотреть более подходящий алгоритм создания хэш-функций, удовлетворяемый несколькими параметрами, такими как быстродействие, большое количество вариаций составленных хэш-функций, а также надёжность выбранного алгоритма. В данном случае, надёжность можно определить по рекомендации специализированных институтов и результатам проводимых конкурсов, а быстродействие и большое количество вариаций должны быть уравновешены.

На выбор были представлены несколько алгоритмов хеширования в свободном доступе:

– серия алгоритмов хэш-функции, принадлежащие семейству MD, разработанные Рональдом Л. Ривестом, что используются преимущественно для хранения паролей или создания уникальных криптографических ключей и ЭЦП. Несмотря на свою известность, последний алгоритм семейства MD – MD5 [1], созданный ещё в 1991 году, достаточно уязвим к некоторым атакам и не желателен для использования. Криптографический алгоритм представлен на рис. 2.

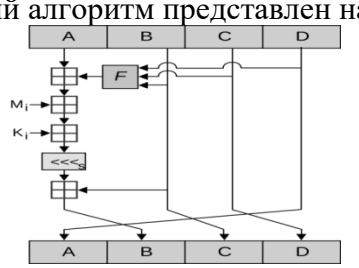

Рис. 2 – Криптографический алгоритм MD5
Fig. 2 – Cryptographic algorithm MD5

– алгоритм CRC, так называемый полином циклической проверки излишков, который является алгоритмом нахождения контрольной суммы, имеет большое количество вариаций и применяется для обнаружения ошибок в потоке информации. Данный алгоритм не является криптографическим и несмотря на его быстродействие, он не подходит для создания безопасных хэш-функций. Основной функцией алгоритма является обнаружение ошибок при передаче данных [2]. Алгоритм работы CRC в приложениях заключается в следующем – при передаче сообщения, получатель принимает не только само сообщение, но и контрольную сумму, которую затем алгоритм на стороне получателя получает из полученного

сообщения и в случае расхождения контрольных сумм можно удостовериться, что сообщение либо повреждено, либо изменено намерено.

- алгоритм Whirlpool – разработанный в 2000 году Винсентом Рэймоном и Пауло Баррето на входе принимает строку в 2^{256} бит, а на выходе выдаёт 512 бит. Считается криптографически стойким, широко применяемым алгоритмом хэш-функции. Несмотря на надёжность, для используемого программного комплекса данный алгоритм будет излишним.
- алгоритм MurmurHash считается довольно быстрой и эффективной, пусть и не является криптографической хэш-функцией, но всё же широко используется в различных целях, хоть и не рекомендован для использования в целях безопасности из-за уязвимости к коллизионным атакам.
- алгоритмы BLAKE2, доступны в сразу двух версиях, BLAKE2b и BLAKE2s, где первый алгоритм оптимизирован для 64-разрядных платформ и выдаёт хэш-функции до 512 бит, в то время как вторая версия оптимизирована для 32-разрядных платформ и может выдавать хэш-функции до 256 бит. Сама по себе хэш-функция BLAKE2 разработана для обеспечения скорости и безопасности, что сравнимо с улучшением популярного алгоритма SHA-3, а также активно применяется в приложениях, требующих высокоскоростного хэширования, как, например, создание криптовалюты. Алгоритм BLAKE представлен на рис. 3.

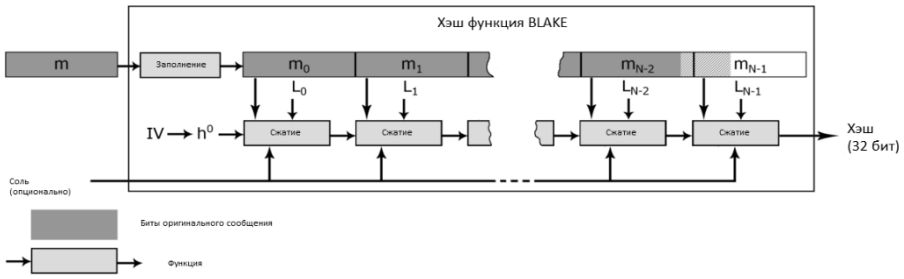

Рис. 3 – Криптографический алгоритм BLAKE
Fig. 3 – Cryptographic algorithm BLAKE

- алгоритмы семейства SHA, так называемый безопасный алгоритм хэширования – являются семейством криптографических хэш-функций, принимающих сообщение и вычисляющих хэш-код фиксированной длины. Разработаны АНБ США, а затем прошедшие публикацию NIST (Национальный институт стандартов и технологий).

По сравнению с другими алгоритмами, существует четыре «поколения» алгоритмов хэш-функций, принадлежащих семейству SHA: SHA-0 – исходная версия 160-битной хэш-функции, опубликованной в 1993 году; SHA-1 – исправленная версия предыдущего алгоритма, опубликованная в 1995 году, но в нём также была найдена уязвимость, в результате чего к нему потеряли доверие; SHA-2 – набор криптографических хэш-функций, опубликованный в 2001 году и рекомендованный NIST к применению в новых системах, при этом включающий в себя различные модификации, отвечающие за различные размеры ключа; SHA-3 – последняя версия алгоритма семейства SHA, выбранная в 2012 году после проведённого конкурса, при этом поддерживает те же длины хэш-кода, что и SHA-1 и SHA-2, но при этом представляет собой совершенно новый алгоритм, считающийся более безопасным. Несмотря на существование более новых алгоритмов, сейчас также активно используются разновидности SHA-2, под названием SHA-256 [3] и SHA-512, отличающиеся размером выходной строки, соответственно на 32 бита и 64 бита.

Общая схема криптографического алгоритма SHA-256 представлена на рис. 4.

В данном программном модуле используется SHA-256, как наиболее оптимальный, из-за его достаточно высокой скорости, стойкости и относительно небольших размерах. Содержится в стандартной библиотеке для языка программирования «Python», под назва-

нием «hashlib». Сжатие файла – это процесс, в ходе которого файл большого размера преобразуется в файл меньшего размера.

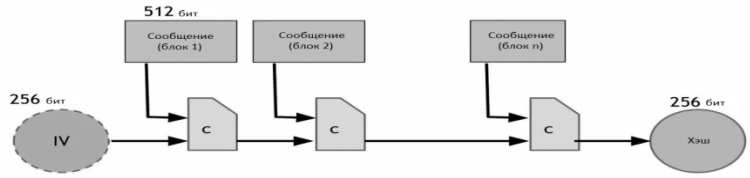

Рис. 4 – Криптографический алгоритм SHA-256
Fig. 4 – Cryptographic algorithm SHA-256

Существует два способа сжатия – с потерями и без потерь. Сжатие с потерями, как понятно из названия, это – сжатие, в результате которого часть информации, в основном незначительная, теряется. Данным методом сжимают аудио, видео и графические файлы, где потери информации не являются существенными. Сжатие без потерь, соответственно, сжимает файл без потери информации, благодаря чему информация файла не теряется. В данном программном комплексе используется сжатие без потерь информации.

Несмотря на свою многочисленность, алгоритмы сжатия не могут похвастаться таким же разнообразием, и часто используемые алгоритмы представлены в основном семейством LZ и его многочисленными модификациями. Например, такие как LZ77, строение которого разобрано в статье [4], или же LZ78, или LZW. Однако потребность сжимать всё большие объёмы данных требуют разработки новых и новых алгоритмов сжатия данный вопрос рассматривается в статьях [5] и [6], но несмотря на это все они до сих пор основываются на RLE, чей алгоритм представлен в виде примера с числами на рис. 5.

Рис. 5 – Принцип алгоритма RLE на примере последовательности чисел
Fig. 5 – The principle of the RLE algorithm using a sequence of numbers as an example

В данном программном комплексе «BadCoreGuard» используется встроенный алгоритм из стандартного набора библиотек языка программирования «Python», под названием «zlib», алгоритм которой также основан на RLE.

Существует несколько видов резервного копирования, у каждого из которых имеются свои недостатки и особенности, что могут использоваться повсеместно или в каких-нибудь определённых условиях. Различные разработки, а, следовательно, и компании, предпочитают определённые механизмы создания резервных копий, или же алгоритмов восстановления данных. Можно выделить пять наиболее популярных механизмов [7], среди которых выделяются традиционные [8], а также новые алгоритмы.

Полное резервное копирование – это алгоритм, который создаёт абсолютную копию исходных файлов, из-за чего считается одним из лучших для защиты данных, благодаря простоте и относительно высокой скорости восстановления. Однако, из-за больших размеров резервных копий восстановление данных занимает довольно много времени, кроме того, сохранение предыдущих образов требует соответствующих объёмов хранилищ. Также стоит отметить, что создание полной резервной копии создаёт сильные нагрузки на сетевой трафик в случае облачного хранилища. Кроме того, в случае повреждения резервной копии, ущерб будет нанесён всем файлам. Общий алгоритм полного резервного копирования представлен на рис. 6.

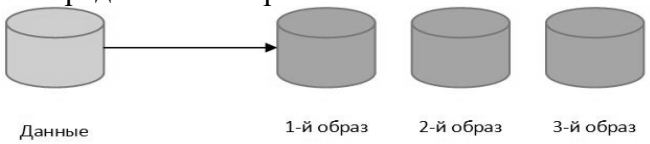

Рис. 6 – Алгоритм полного резервного копирования
Fig. 6 – Full backup algorithm

Инкрементальное резервное копирование – данный алгоритм (представленный на рис. 7) сокращает нагрузку на сеть и затрачиваемое количество времени, необходимое для создания резервной копии. В основе алгоритма лежит создание полного образа данных, после чего в него записываются только те блоки данных, которые изменились с момента создания резервной копии.

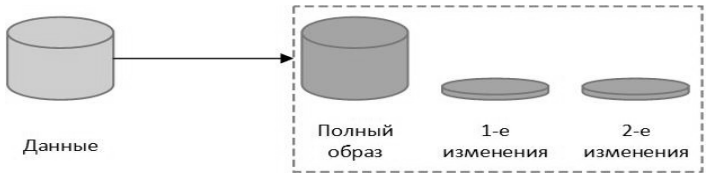

Рис. 7 – Алгоритм инкрементального резервного копирования
Fig. 7 – Incremental backup algorithm

Таким образом, сначала создаётся полный образ, а затем в него добавляются изменения, называемые инкрементами, прошедшие с последнего копирования, и так до создания очередного полного образа, что определяется политикой хранения резервных копий. Однако, у данного способа есть свои удобства и недостатки – несмотря на высокую скорость, по сравнению с полным резервным копированием, а также меньшим занимаемым пространством хранилища, процесс восстановления будет медленным из-за необходимости вернуться к изначальной резервной копии через каждый инкремент, что подразумевает в случае повреждения цепочки инкрементов восстановление данных может быть невозможно.

Дифференциальное резервное копирование – этот алгоритм является промежуточным решением между инкрементальным и полным алгоритмами резервного копирования. Аналогично инкрементальному, началом алгоритма является создание полной резервной копии, а затем внесение в неё изменений, но изменения вносятся не относительно последнего копирования, то есть с момента каждого инкремента, а относительно первоначального образа, что показано на рис. 8.

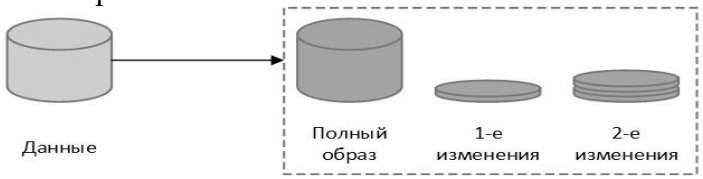

Рис. 8 – Алгоритм дифференциального резервного копирования
Fig. 8 – Differential backup algorithm

Благодаря такому подходу, данный алгоритм по сравнение с инкрементальным быстрее восстанавливается данные, но медленнее создаёт сами образы. Однако со временем, инкременты будут требовать всё больше и больше места и времени, в иных случаях превосходя даже полное резервное копирование.

Реверсивное инкрементальное резервное копирование – модификация инкрементального алгоритма, заключающийся в внедрении инкрементов к первоначальному полному образу, при этом сохраняя все замещаемые блоки, что позволяет восстановить предыдущие версии сохранённых данных. Из преимуществ можно выделить высокую скорость восстановления данных из резервной копии, так как в данном алгоритме используется последний полный образ данных. Данный алгоритм представлен на рис. 9.

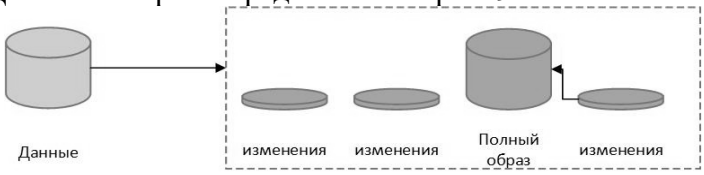

Рис. 9 – Алгоритм реверсивного инкрементального резервного копирования
Fig. 9 – Algorithm of reverse incremental backup

Синтетическое полное резервное копирование – является модификацией алгоритма реверсивного инкрементального резервного копирования, представлен на рис.10. Созда-

ние синтетической полной резервной копии аналогично алгоритму реверсивного инкрементального, однако инкрементальные образы со временем объединяются и применяются к имеющемуся первоначальному первому образу, тем самым формируя резервную копию в качестве нового образа. Данный метод включает в себя все преимущества стандартных алгоритмов создания резервных копий, то есть обладает низкими требованиями к хранилищу, что позволяет более эффективно управлять пространством, создаёт низкие нагрузки на сеть, а также обеспечивает высокую скорость создания резервных копий и восстановления данных из формируемого образа. У данного алгоритма также имеется довольно серьёзный недостаток – высокая нагрузка на сервер резервного копирования, из-за чего его содержание может обходиться дороже.

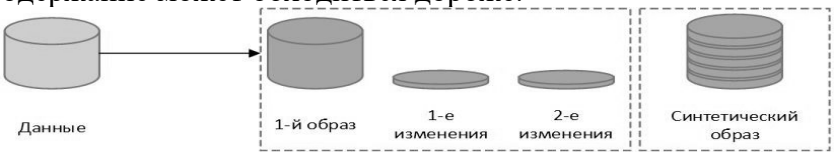

Рис. 10 – Алгоритм синтетического полного резервного копирования
Fig. 10 – Synthetic full backup algorithm

Для разработки программного модуля был выбран реверсивный инкрементальный алгоритм резервного копирования, как наиболее подходящий для разработанного алгоритма и более удобный в использовании.

Обсуждение результатов. При разработке программного средства был создан новый модифицированный алгоритм, использующий в своей основе три связанных друг с другом алгоритма, отвечающие за работу программы – сканирования, настройки правил сканирования, а также восстановления файлов. Алгоритм сканирования представлен в виде блок-схемы на рис. 11.

Суть данного алгоритма заключается в предоставлении четырёх разных режимов сканирования выбранной области и формировании хэш-таблицы просканированных файлов, чтобы в дальнейшем данные файлы можно было проверить на наличие несанкционированных изменений. Алгоритм имеет четыре режима сканирования файлов, что может использовать для более тонкой настройки сканирования файлов и создания таблиц хэш-кодов.

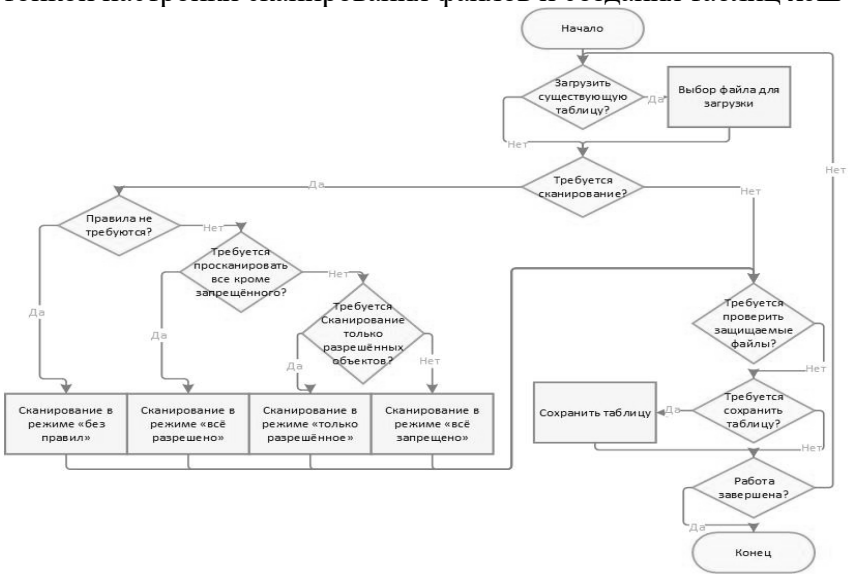

Рис. 11 – Блок-схема, описывающая алгоритм сканирования файлов
Fig. 11 – Flowchart describing the file scanning algorithm

Программная реализация представлена в виде графического интерфейса на рис. 12. Первая вкладка, «Работа с хэш-таблицей», отвечает за сканирование файлов и директорий, а также работы с таблицей хэш-кодов и их проверке. При запуске любого режима, кроме режима «Только разрешенные», будет открыто окно выбора, необходимое для задания сканируемой директории. Запуск в режиме «Без правил» стоит по умолчанию, даже если

при запуске сканирования не отмечен ни один режим. В данном режиме сканируются все файлы в директории, а также все директории внутри неё и файлы, входящие в них.

В режиме «Всё разрешено» сканируется вся директория, её файлы и все директории в неё входящие, но в случае если в параметрах проверки присутствуют какие-либо запреты на сканирование файлов или директорий, то программа проигнорирует их.

При запуске в режиме «Всё запрещено» программа игнорирует все директории и файлы кроме тех, что помечены как разрешенные. В случае если в директории, не отмеченные как разрешённые, размещён файл или директория, помеченные как разрешённые, то в таком случае программа не просканирует их. При сканировании в режиме «Только разрешенные» программа перейдёт по сохранённым путям файлов, помеченных как разрешенных, и просканирует их.

Рис. 12 – Программная реализация алгоритма сканирования файлов
Fig. 12 – Software implementation of the file scanning algorithm

Алгоритм настройки правил сканирования файлов представлен в виде блок-схемы на рис. 13. Данный алгоритм позволяет провести настройку, для выбора файлов и директорий, что будут подлежать сканированию, а какие наоборот – не будут подвергаться процедуре сканирования.

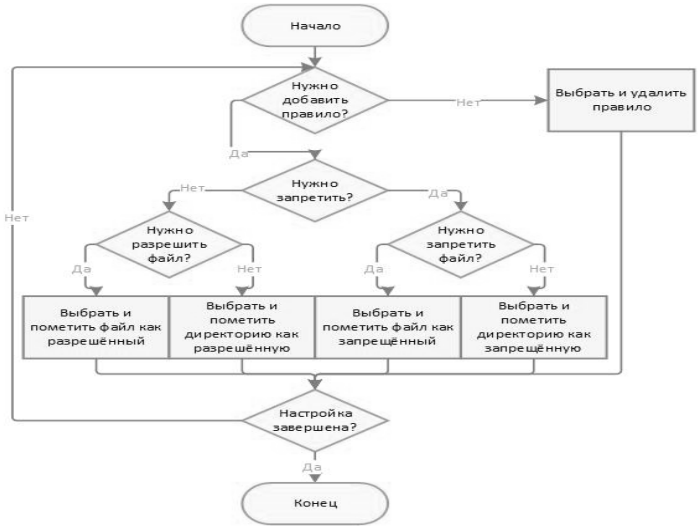

Рис. 13 – Блок-схема, описывающая алгоритм настройки сканирования файлов
Fig. 13 – Flowchart describing the file scanning setup algorithm

Программная реализация представлена в виде графического интерфейса на рис. 14.

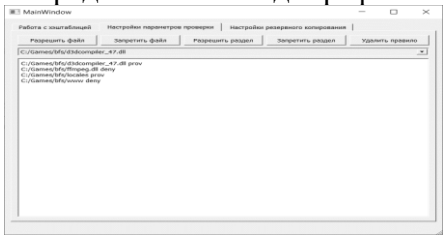

Рис. 14 – Программная реализация алгоритма настройки сканирования файлов
Fig. 14 – Software implementation of the file scanning setup algorithm

Данная вкладка «Настройки параметров проверки» имеет пять функций, определяющих правила сканирования. Создание правила на разрешения доступа к файлу, а также его

запрет. Создание правила разрешающего и запрещающего доступа к директории. Также присутствует функция, позволяющая удалить добавленное правило.

Блок-схема алгоритма, отвечающего за восстановление файлов представлена на рис. 15. Данный алгоритм сохраняет резервные копии файлов в обозначенной директории, а также восстанавливает их в случае, если файл был не санкционированно изменён.

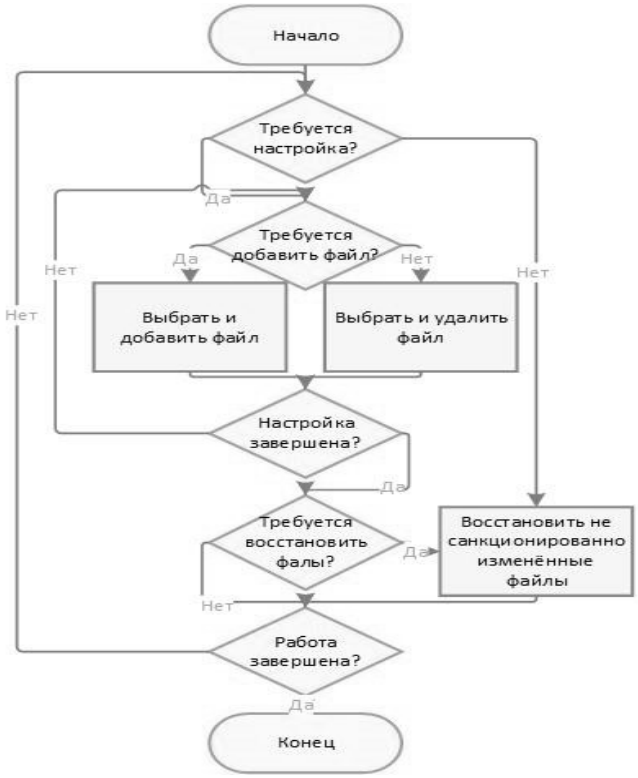

Рис. 15 – Блок-схема, описывающая алгоритм восстановления файлов
Fig. 15 – Flowchart describing the file recovery algorithm

Программная реализация представлена в виде графического интерфейса на рис. 16. Данная вкладка «Настройки резервного копирования» позволяет настраивать директорию для сохранения резервных копий файлов в формате «.amongus», добавить файл и создать его резервную копию, а также восстановить его или же удалить, используя соответствующие функции.

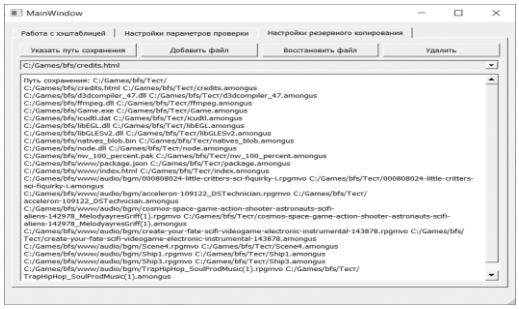

Рис. 16 – Программная реализация алгоритма восстановления файлов
Fig. 16 – Software implementation of the file recovery algorithm

Использование трёх алгоритмов сканирования, настройки правил и восстановления файлов позволяет создать крайне удобный инструмент для создания таблиц хэш-кодов, проверки не санкционированно изменённых файлов, которые можно восстановить из резервной копии. В настоящий момент существуют и другие программные решения резервного копирования и восстановления данных, которые рассматривались в качестве аналогов – пять программных решений, позволяющих провести резервное копирование, то есть восстановить повреждённые файлы.

Рассмотрим первый вариант – Easis Free Drive Cloning. Данная утилита содержит несколько функций, такие как, создание образа данных, восстановление данных из созданного образа и клонирование дисков. Графический интерфейс данного программного средства прост и удобен для обычных пользователей. Также этот инструмент создаёт резервную копию свободных секторов диска вместе с секторами, имеющими фактические данные. Однако, по сравнению с разработанным программным средством не обладает возможностью формирования таблиц хэш-кодов и настройкой сканирования файлов или директорий.

Следующим программным средством является Paragon Backup and Recovery – утилита, позволяющая восстанавливать файлы, директории и диски из образа и отдельных инкрементных точек сохранения. Может сжать и разделить данные, чтобы сэкономить место на диске. Несмотря на предоставляемые преимущества, данный инструмент не имеет возможности создания таблиц хэш-кодов и настройки сканирования.

Программное средство Action Backup обладает интуитивно понятный интерфейс и наделена широкими функциональными возможностями. Данный инструмент имеет возможность создавать (дублировать) резервные копии на множестве локальных и сетевых ресурсах, создавать образ диска, выполнять круглосуточное резервное копирование в режиме службы WINDOWS, отправлять отчеты о выполненных работах на E-Mail. Несмотря на предоставляемые преимущества, всё также отсутствует формирование хэш-таблиц и настройка параметров сканирования.

Утилита Comodo Backup спроектирована и разработана как наиболее подходящий вариант для обычных пользователей. Этот инструмент позволяет копировать документы, мультимедийные файлы, записи реестра, электронные письма, историю чата, все другие журналы и файлы. Также имеется возможность создать резервную копию данных в локальном или облачном хранилище. Формирование хэш-таблиц и настройка сканирования отсутствует.

EASEUS Todo Backup – является программным средством, созданным для решения задач резервного копирования, а также имеющее множество функций. Он предлагает три способа резервного копирования файлов – полный, дифференциальный и дополнительный. Кроме того, утилита предлагает создание образов с возможностью создания загрузочного диска на основе Linux. Имеется возможность создавать собственные резервные копии в любое время и даже планировать процесс резервного копирования. Однако, данная утилита не поддерживает отдельное сохранение файлов, так как работает с полными образами, а также не имеет возможности формирования таблиц хэш-кодов и настройки сканирования. Проведённый анализ разработанного программного средства с существующими аналогами был сформирован в табл.1.

Таблица 1. Сравнительный анализ программной реализации с аналогами
Table 1. Comparative analysis of software implementation with analogues

Название **Title**	**Формирование хеш-таблиц** **Generating Hash tables**	**Настройка сканирования** **Scan settings**	**Сохранение файлов** **Saving files**	**Сохранение образа диска** **Saving a disk image**
Easis Free Drive Cloning	-	-	-	+
Paragon Backup and Recovery	-	-	+	+
Action Backup	-	-	+	+
Comodo Backup	-	-	+	+
EASEUS Todo Backup	-	-	-	+
Bad Core Guard **(разработанное средство)**	+	+	+	+

В результате проведённых сравнений, были выявлены недостатки разработанного программного средства и составлен план его дальнейших модификаций для наиболее

удобной и практичной работы. После проведения сравнения функционала, было проведено тестирование разработанного программного средства, представленное на рис. 17 и 18, представляющих графики, показывающих эффективность работы программы относительно размера файла и времени, затраченного на работу алгоритмов.

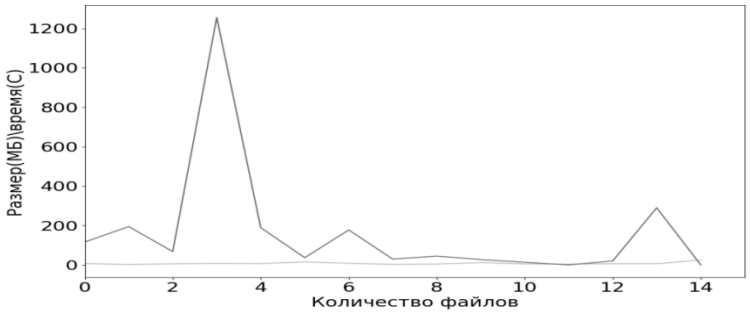

Рис. 17 – График тестирования алгоритма создания резервной копии
Fig.17 – Backup creation algorithm testing schedule

На графике, представленном на рис.17 показана зависимость затраченного времени, относительно размеров самого файла для создания резервной копии. Четырнадцать файлов с различными размерами от не превышающих единицу и свыше тысячи мегабайт, показывают незначительные расхождения в скорости формирования резервных копий, не превышающих половину минуты. В результате полученных данных можно отметить, что время формирования резервной копии файла практически не зависит от его размера.

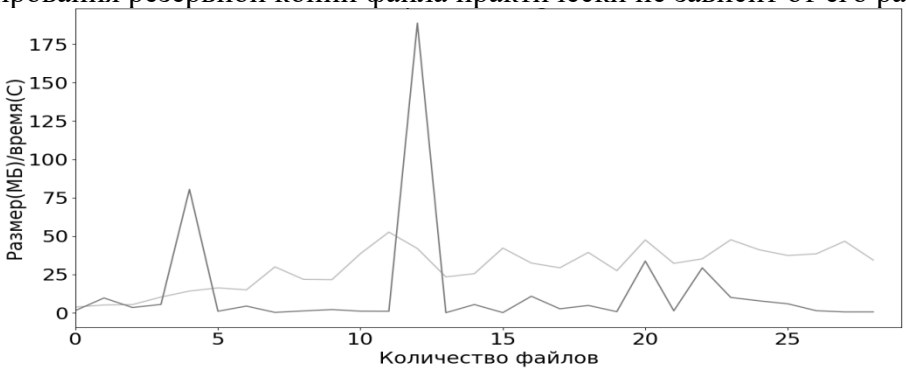

Рис. 18 – График тестирования алгоритма формирования таблицы хэш-кодов
Fig. 18 – Graph of testing the algorithm for generating a table of hash codes

На графике, представленном на рис. 18 также показана зависимость затраченного времени, относительно размеров самого файла для формирования таблицы хэш-кодов. Двадцать девять файлов с различными размерами от не превышающих единицу и свыше сто пятидесяти мегабайт, показывают незначительные расхождения в скорости формирования таблицы хэш-кодов, не превышающих минуту. Стоит отметить, что значительную часть времени занимает сам алгоритм формирования хэш-кода, который также нуждается в оптимизации для более быстрой работы.

В результате полученных данных можно отметить, что время формирования резервной копии файла в незначительно зависит от его размера. По результатам тестирования скорость работы алгоритма резервного копирования не зависит от размера файла, а скорость формирования хэш-таблицы зависит от объёма сканируемой информации незначительно.

Для разработки программного средства был выбран язык Python, так как данный язык зарекомендовал себя одним из самых популярных и широко используемых в разработке пользовательских приложений. Вместе с этим, также была выбрана среда программирования – Visual Studio Code от компании Microsoft [9].

Для разработки графической оболочки, использовалось программное средство QtDesigner – кроссплатформенная свободная среда для разработки графических интерфейсов [10,11]. Python – это интерпретируемый программный язык высокого уровня,

часто используемый множеством разработчиков для создания пользовательских приложений и обладает автоматическим управлением памяти, позволяющим повысить переносимость написанных на данном языке программных средств [12, 13]. Для создания программного средства использовалась версия Python 3.11.

Вывод. В результате проведенного исследования все поставленные цели были достигнуты, а именно:

- проведено исследование и выбор существующих алгоритмов хэширования;
- произведен анализ существующих алгоритмов сжатия и восстановления данных; разработана реализация программного средства;
- проведен сравнительный анализ реализованного средства с аналогами и модификациями.

В результате проведённых тестов, программное средство обнаружило изменённые и повреждённые файлы. Тестирование прошло успешно, тем самым показывая эффективность данного программного средства для восстановления и защиты файлов от повреждений. Результаты исследования доказывают, что каждый пользователь может защитить свои личные файлы от угроз при использовании данного программного средства.

Предложенный механизм восстановления данных является современным решением, обеспечивающим сохранность личных файлов в случае их повреждения.

Для будущего улучшения программного средства были выделены основные задачи, что будут рассмотрены при его обновлении: расширение функционала программного средства; оптимизация программного кода для достижения большего быстродействия; обновление и улучшение модулей программного средства; добавление функций копирования образа диска.

Библиографический список:

1. Телегин В.А. Анализ криптографической стойкости модифицированного алгоритма md5. Universum: технические науки, № 9–2 (114), 2023, с. 16–20.
2. Клименко С.В., Яковлев В.В., Благовещенская Е.А. Исследование реализаций алгоритмов контрольной суммы CRC32" Известия Петербургского университета путей сообщения, т. 15, № 3, 2018, с. 471–477.
3. Астахов С.В., Вариханов Д.И. Вычислитель хеш-функции SHA-256. Политехнический молодежный журнал, 2023, № 08 (85). URL: http://dx.doi.org/10.18698/2541-8009-2023-8-924 (дата обращения 3.10.2023).
4. Эрдман А.А. Реализация алгоритма сжатия данных Лемпеля–Зива LZ77 на языке программирования Python. Постулат. – 2023. – №. 1.
5. Андриенко И.С. Разработка алгоритма сжатия данных RLE на языке программирования Python. Постулат. – 2023. – №. 1.
6. Звайгзне А.Ю. Создание алгоритма сжатия текстовых данных на языке программирования Python. Постулат. – 2023. – №. 1.
7. Чумбуридзе Я.А. Основные методы резервного копирования для обеспечения безопасности информации. Международный журнал гуманитарных и естественных наук, № 3-2 (78), 2023, с. 57–60.
8. Бопп В.А. Особенности выбора систем резервного копирования. Известия Тульского государственного университета. Технические науки, № 10, 2019, с. 297–300.
9. Официальный сайт программной среды Visual Code. URL: https://code.visualstudio.com/docs, дата обращения 12.11.2023.
10. Официальный сайт кроссплатформенной среды для разработки графической оболочки QtDesigner. URL: https://doc.qt.io/qt-6/qtdesigner-manual.html, дата обращения 16.11.2023.
11. Официальный сайт программного языка Python. URL:https://www.python.org/doc/, дата обращения 19.11.2023.
12. Импортозамещающие технологии обеспечения информационной безопасности и защиты данных : учебное пособие /Д.А. Короченцев, Л.В. Черкесова, Е.А. Ревякина [и др.]. — Ростов-на-Дону : Донской ГТУ, 2021. — 335 с.
13. Development of a real-time document approval system Cherckesova L., Boldyrikhin N., Revyakina E., Safaryan O., Yengibaryan I. В сборнике: E3S Web of Conferences. 14th International Scientific and Practical Conference on State and Prospects for the Development of Agribusiness, INTERAGROMASH 2021. Rostov-on-Don, 2021. С. 08047.

References:

1. Telegin V.A. Analysis of cryptographic resistance of the modified md5 algorithm. *Universum: technical sciences*, 2023; 9-2 (114):16-20. (In Russ).
2. Klimenko S.V., Yakovlev V.V., Blagoveshchenskaya E.A. Study of implementations of CRC32 checksum algorithms. *Bulletin of the Petersburg State University of Railway Engineering*, 2018; 15(3): 471-477. (In Russ).
3. Astakhov S.V., Varihanov D.I. SHA-256 hash function calculator. *Polytechnic Youth Journal*, 2023; 08 (85). URL: http://dx.doi.org/10.18698/2541-8009-2023-8-924 (date of access 3.10.2023). (In Russ).
4. Erdman A.A. "Implementation of the Lempel-Ziv LZ77 data compression algorithm in the Python programming language." *Postulate*. 2023;1. (In Russ).
5. Andrienko I.S. "Development of the RLE data compression algorithm in the Python programming language." *Postulate.* 2023;1. (In Russ).
6. Zvaigzne A.Yu. "Creation of a text data compression algorithm in the Python programming language." Postulate. 2023;1. (In Russ).
7. Chumburidze Ya.A. "Basic backup methods to ensure information security" *International Journal of Humanities and Natural Sciences*, 2023; 3-2 (78):57–60. (In Russ).
8. Bopp V. A. "Features of choosing backup systems" *Bulletin of Tula State University. Technical sciences*, 2019; 10:297–300. (In Russ).
9. Official website of the Visual Code software environment. URL: https://code.visualstudio.com/docs, accessed 11/12/2023.
10. Official website of the cross-platform environment for developing the QtDesigner graphical shell. URL: https://doc.qt.io/qt-6/qtdesigner-manual.html, accessed 11/16/2023.
11. Official website of the Python programming language. URL: https://www.python.org/doc/,accessed 19.11.2023.
12. Import-substituting technologies for ensuring information security and data protection: a tutorial / D.A. Korochentsev, L.V. Cherkesova, E.A. Revyakina [et al.]. Rostov-on-Don: Donskoy GTU, 2021;335(In Russ).
13. Development of a real-time document approval system Cherckesova L., Boldyrikhin N., Revyakina E., Safaryan O., Yengibaryan I. In the collection: E3S Web of Conferences. 14th Inter-national Scientific and Practical Conference on State and Prospects for the Development of Agribusiness, INTERAGROMASH 2021. Rostov-on-Don, 2021: 08047.

Сведения об авторах:

Черкесова Лариса Владимировна, доктор физико-математических наук, профессор, профессор, кафедра «Кибербезопасность информационных систем»; chia2002@inbox.ru

Савельев Василий Александрович, кандидат физико-математических наук, доцент, доцент, кафедра «Кибербезопасность информационных систем»; vasav.rnd@mail.ru

Ревякина Елена Александровна, кандидат технических наук, доцент, доцент, кафедра «Кибербезопасность информационных систем»; Revyelena@yandex.ru

Полулях Анатолий Русланович, студент, кафедра «Кибербезопасность информационных систем»; tolik241510@gmail.com

Семенцов Максим Александрович, студент, кафедра «Кибербезопасность информационных систем»; maxim00990@yandex.ru.

Information about authors:

Larisa V. Cherkesova, Dr. Sci. (Physics and Mathematics), Prof., Prof., Department «Cybersecurity of information Systems»; chia2002@inbox.ru

Vasily A. Savel'ev, Cand. Sci. (Physics and Mathematics), Assoc. Prof., Assoc. Prof., Department «Cybersecurity of information Systems»; vasav.rnd@mail.ru

Elena A. Revyakina, Cand. Sci. (Physics and Mathematics), Assoc. Prof., Assoc. Prof., Department «Cybersecurity of information systems»; Revye-lena@yandex.ru

Anatoly R. Polulyakh, Student, Department «Cybersecurity of information Systems»; tolik241510@gmail.com

Maxim A. Semencov, Student, Department «Cybersecurity of information systems»; maxim00990@yandex.ru.

Конфликт интересов/Conflict of interest.

Авторы заявляют об отсутствии конфликта интересов/The authors declare no conflict of interest.

Поступила в редакцию/ Received 15.08.2024.

Одобрена после рецензирования/ Reviced 30.09.2024.

Принята в печать/ Accepted for publication 29.12.2024.