

Разработка модуля верификации целостности файлов на основе хеш-функции SHA-256 на языке Python

Гиренко Александр Евгеньевич

Приамурский государственный университет им.Шолом-Алейхема

Студент

Аннотация

В статье рассматривается задача контроля целостности файлов в вычислительных системах. Предлагается программный модуль на языке Python, реализующий алгоритм SHA-256 для вычисления хеш-сумм файлов. Модуль позволяет сохранять эталонное состояние файловой системы и при повторном запуске выявлять добавленные, удалённые и изменённые файлы. Разработанное решение может применяться для обнаружения несанкционированных изменений, аудита файловых систем и контроля резервных копий. Экспериментальная проверка проведена на тестовом наборе из двух файлов: после внесения изменений модуль корректно идентифицировал 100% модификаций.

Ключевые слова: хеш-функция, SHA-256, целостность данных, Python, верификация файлов, информационная безопасность.

Development of a file integrity verification module based on the SHA-256 hash function in Python

Girenko Aleksandr Evgen'evich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article discusses the problem of file integrity control in computing systems. A software module in Python is proposed that implements the SHA-256 algorithm for calculating file hash sums. The module allows saving the reference state of the file system and, upon restart, detecting added, deleted, and modified files. The developed solution can be used for detecting unauthorized changes, auditing file systems, and monitoring backups. Experimental verification on a test set of two files showed that after making changes, the module correctly identified 100% of modifications.

Keywords: hash function, SHA-256, data integrity, Python, file verification, information security.

1 Введение

1.1 Актуальность

В современном цифровом мире проблема обеспечения целостности данных становится критически важной. Ежедневно пользователи и организации сталкиваются с угрозами несанкционированного изменения файлов: вирусные атаки, подмена системных библиотек, ошибки при резервном копировании, случайные или преднамеренные правки документов. Традиционные методы контроля (сравнение даты изменения, размера файлов) недостаточно надёжны, так как эти метаданные могут быть легко подделаны злоумышленником. Более надёжным подходом является использование криптографических хеш-функций, которые позволяют получить уникальный «цифровой отпечаток» содержимого файла.

1.2 Обзор исследований

В работе А.Р. Полулях и К.И. Дубиковой рассматриваются хеш-функции и их виды, анализируется проблема коллизий и их влияние на надежность хеширования [1]. К.К. Тинг, С.К.Л. Юен, К.Х. Ли и Ф.Х.В. Леонг представляют разработку SHA-256 процессора на базе FPGA, описывая аппаратную реализацию криптографического алгоритма [2]. Д.А. Коростелев, Д.В. Логвинов и С.С. Савкин разрабатывают программу для демонстрации и анализа работы электронной цифровой подписи RSA с хешированием SHA-256, реализованную на языке Python [3]. Е.Н. Петров, М.А. Симонова, Р.А. Мязина, А.В. Ларчиков и В.В. Кокин представляют программу для обработки и верификации конфигурационных файлов сетевого оборудования, описывая возможности автоматизации проверки целостности данных [4]. А.З. Кашежев проводит сравнение хэш-сумм на основе алгоритмов SHA-1, SHA-256 и MD5 с использованием Python, анализируя их криптографическую стойкость и применимость для обеспечения целостности файлов [5].

1.3 Цель исследования

Целью исследования является разработка программного модуля на языке Python для контроля целостности файлов с использованием хеш-функции SHA-256, обеспечивающего следующие функции:

1. вычисление хеш-сумм всех файлов в заданной директории (с обходом подпапок);
2. сохранение эталонного состояния в JSON-файл с временной меткой;
3. детектирование добавленных, удалённых и изменённых файлов при повторном запуске;
4. формирование наглядного отчёта на экране и сохранение его в файл.

2 Метод и алгоритм

2.1 Обоснование выбора хеш-функции

Для верификации целостности файлов используется хеш-функция SHA-256 (Secure Hash Algorithm 256-bit). Выбор обусловлен следующими характеристиками (Табл. 1).

Таблица 1 – Сравнительная характеристика хеш-функций анализ пропусков и методы их заполнения

Характеристика	MD5	SHA-1	SHA-256
Длина хеша (бит)	128	160	256
Длина хеша (hex)	32	40	64
Коллизионная устойчивость	Скомпрометирована	Ослаблена	Высокая
Рекомендация к использованию	Не рекомендуется	Не рекомендуется	Рекомендуется

SHA-256 обладает высокой коллизионной устойчивостью (отсутствие известных практических атак нахождения коллизий) и поддерживается встроенным модулем hashlib языка Python.

2.2 Алгоритм работы модуля

Предлагаемый алгоритм верификации целостности файлов состоит из следующих этапов:

1. Сканирование — рекурсивный обход всех файлов в указанной директории (исключая файл состояния).
2. Вычисление хеша — для каждого файла считываются данные блоками по 4096 байт, и вычисляется SHA-256 хеш с помощью hashlib.sha256().
3. Сохранение эталона — при первом запуске словарь сохраняется в JSON-файл с временной меткой.
4. Сравнение — при повторном запуске вычисляются текущие хеши и сравниваются с сохранёнными.
5. Формирование отчёта — определяются три категории изменений: добавленные, удалённые и изменённые файлы.

Блок-схема алгоритма представлена на Рисунке 1.

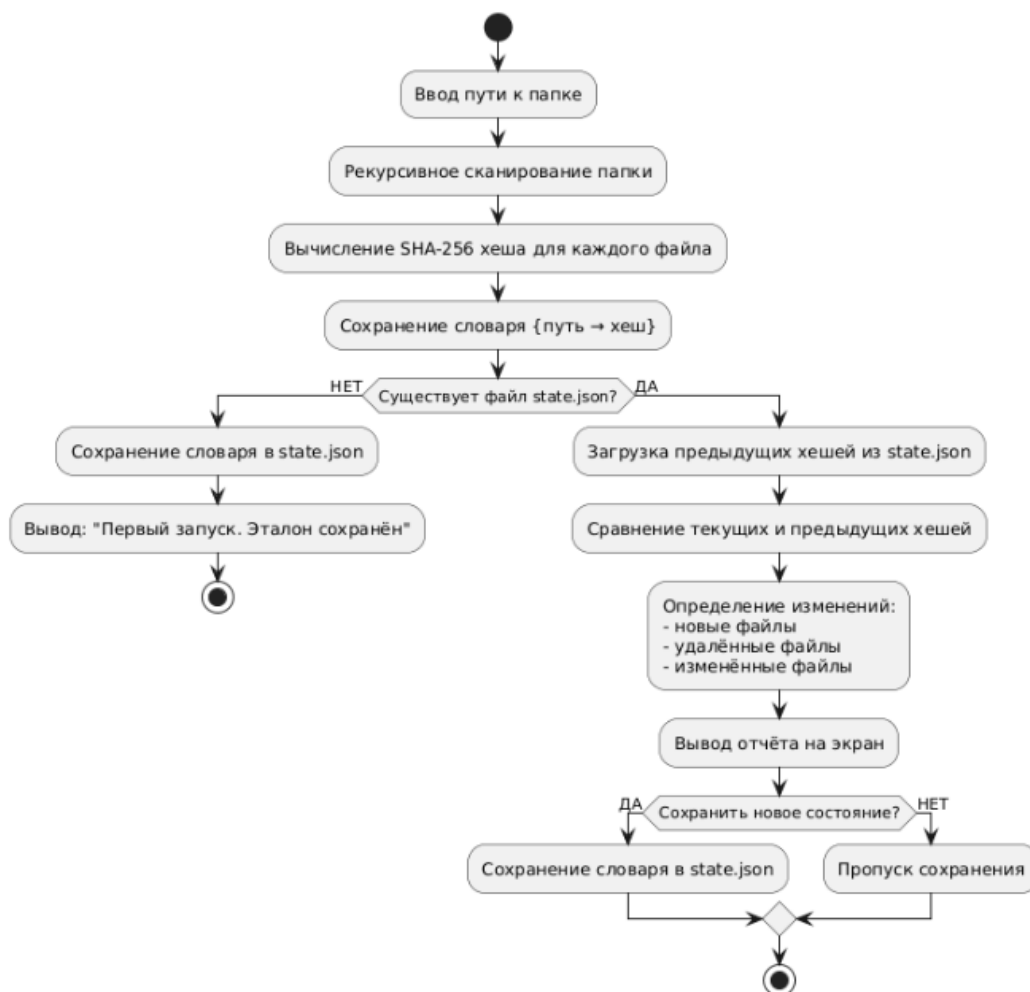


Рисунок 1 — Блок-схема алгоритма верификации целостности файлов

2.3 Программная реализация

Модуль реализован на языке Python версии 3.8+ с использованием только стандартной библиотеки.

Вычисление хеша файла. Для получения уникального идентификатора содержимого файла реализована функция `get_file_hash()`, которая принимает путь к файлу и возвращает его SHA-256 хеш в шестнадцатеричном формате. Функция читает файл блоками по 4096 байт, что позволяет обрабатывать файлы любого размера без избыточного потребления оперативной памяти. Листинг функции представлен на Рисунке 2.

```
def get_file_hash(filepath): 1 usage
    """Вычисляет SHA-256 хеш файла"""
    sha256 = hashlib.sha256()
    with open(filepath, 'rb') as f:
        while True:
            data = f.read(4096)
            if not data:
                break
            sha256.update(data)
    return sha256.hexdigest()
```

Рисунок 2 — Листинг функции вычисления хеша файла

Сканирование папки. Для рекурсивного обхода файловой системы реализована функция `scan_folder()`, которая принимает путь к корневой папке и возвращает словарь, где ключом является относительный путь к файлу, а значением — его SHA-256 хеш. Обход всех вложенных папок выполняется с помощью метода `rglob('*')`. Файл состояния `state.json` автоматически исключается из сканирования. Листинг функции представлен на Рисунке 3.

```
def scan_folder(folder): 1 usage
    """Сканирует папку и возвращает хеши всех файлов"""
    folder = Path(folder)
    hashes = {}

    for file_path in folder.rglob('*'):
        if file_path.is_file():
            if file_path.name == 'state.json':
                continue
            rel_path = str(file_path.relative_to(folder))
            hashes[rel_path] = get_file_hash(str(file_path))
            print(f" 📄 {rel_path}")

    return hashes
```

Рисунок 3 — Листинг функции сканирования папки

Сохранение состояния. Для сохранения эталонного состояния файловой системы реализована функция `save_state()`, которая записывает полученный словарь хешей в JSON-файл. Структура файла включает временную метку проверки, общее количество файлов и словарь соответствия путей хеш-суммам. Листинг функции представлен на Рисунке 4.

```
def save_state(hashes, filename='state.json'): 2+ usages
    """Сохраняет состояние в JSON файл"""
    data = {
        'date': datetime.now().isoformat(),
        'count': len(hashes),
        'hashes': hashes
    }
    with open(filename, 'w', encoding='utf-8') as f:
        json.dump(data, f, indent=2, ensure_ascii=False)
    print(f"\n📁 Сохранено в {filename}")
```

Рисунок 4 — Листинг функции сохранения состояния

Загрузка состояния. Для восстановления ранее сохранённого эталона реализована функция `load_state()`, которая считывает JSON-файл и возвращает словарь хешей. При отсутствии файла функция возвращает пустой словарь, что соответствует первому запуску программы. Листинг функции представлен на Рисунке 5.

```
def load_state(filename='state.json'): 1 usage
    """Загружает состояние из JSON файла"""
    if not os.path.exists(filename):
        return {}

    with open(filename, 'r', encoding='utf-8') as f:
        data = json.load(f)
    print(f"📁 Загружено от {data.get('date', '?')} (файлов: {data.get('count', 0)})")
    return data.get('hashes', {})
```

Рисунок 5 — Листинг функции загрузки состояния

Сравнение хешей. Для детектирования изменений между текущим и эталонным состоянием реализована функция `compare_hashes()`. Она возвращает словарь с тремя категориями: `added` — новые файлы, `deleted` — удалённые файлы, `modified` — изменённые файлы. Листинг функции представлен на Рисунке 6.

```
def compare(current, previous): 1 usage
    """Сравнивает текущие и предыдущие хеши"""
    curr_set = set(current.keys())
    prev_set = set(previous.keys())

    return {
        'new': list(curr_set - prev_set),
        'deleted': list(prev_set - curr_set),
        'changed': [f for f in (curr_set & prev_set) if current[f] != previous[f]],
        'same': len(curr_set & prev_set) - len([f for f in (curr_set & prev_set) if current[f] != previous[f]])
    }
```

Рисунок 6 — Листинг функции сравнения хешей

Вывод отчёта. Для наглядного представления результатов проверки реализована функция `print_result()`, которая выводит на экран отчёт с разделением по категориям изменений. Листинг функции представлен на Рисунке 7.

```
def print_result(result): 1 usage
    """Выводит результат на экран"""
    print("\n" + "=" * 60)
    print("РЕЗУЛЬТАТ ПРОВЕРКИ")
    print("=" * 60)

    if result['new']:
        print(f"\n📄 НОВЫЕ файлы ({len(result['new'])}):")
        for f in result['new'][:5]:
            print(f"    + {f}")

    if result['deleted']:
        print(f"\n✗ УДАЛЁННЫЕ файлы ({len(result['deleted'])}):")
        for f in result['deleted'][:5]:
            print(f"    - {f}")

    if result['changed']:
        print(f"\n✂ ИЗМЕНЁННЫЕ файлы ({len(result['changed'])}):")
        for f in result['changed'][:5]:
            print(f"    ~ {f}")

    print(f"\n✅ НЕ ИЗМЕНЕНЫ: {result['same']}")

    total = len(result['new']) + len(result['deleted']) + len(result['changed'])
    if total == 0:
        print(f"\n👉 ВСЁ ХОРОШО: Изменений нет")
    else:
        print(f"\n⚠ ВНИМАНИЕ: Обнаружено {total} изменений")

    print("=" * 60)
```

Рисунок 7 — Листинг функции вывода отчёта

Основная функция. Управляющая логика модуля реализована в функции `main()`, которая последовательно выполняет сканирование, загрузку эталона, сравнение и вывод отчёта. Аргумент командной строки позволяет пользователю указать целевую директорию для проверки. Листинг функции представлен на Рисунке 8.

```
def main(): 1 usage
    import sys

    target = sys.argv[1] if len(sys.argv) > 1 else "."

    print("=" * 60)
    print("ВЕРИФИКАЦИЯ ЦЕЛОСТНОСТИ ФАЙЛОВ")
    print("Алгоритм: SHA-256")
    print("=" * 60)
    print(f"Папка: {target}\n")

    print("🔍 Сканирование файлов:")
    current = scan_folder(target)
    print(f"\n📁 Всего файлов: {len(current)}")

    previous = load_state()

    if not previous:
        print("\n🚀 ПЕРВЫЙ ЗАПУСК")
        save_state(current)
        print("\n✅ Готово! При следующем запуске увидите изменения.")
        return

    result = compare(current, previous)
    print_result(result)

    answer = input("\nСохранить новое состояние? (да/нет): ").lower()
    if answer in ('да', 'yes', 'y', 'd'):
        save_state(current)
        print("✅ Сохранено")
    else:
        print("❌ Не сохранено")

if __name__ == "__main__":
    main()
```

Рисунок 8 — Листинг основной функции

3 Экспериментальные результаты

3.1 Условия проведения эксперимента

Экспериментальная проверка модуля проводилась на персональном компьютере со следующими характеристиками:

1. Операционная система: Windows 10;
2. Процессор: Ryzen 5 7500f;
3. Оперативная память: 32 ГБ;
4. Язык реализации: Python 3.14.

Для тестирования создана папка test_files, содержащая два текстовых файла (a.txt и b.txt).

3.2 Результаты первого запуска

При первом запуске модуля с параметром checker.py test_files программа выполнила сканирование папки и сохранение эталонного состояния. Результат представлен на Рисунке 9.

```
=====
ВЕРИФИКАЦИЯ ЦЕЛОСТНОСТИ ФАЙЛОВ
Алгоритм: SHA-256
=====
Папка: test_files

🔍 Сканирование файлов:
  📄 a.txt
  📄 b.txt

📁 Всего файлов: 2
📁 Загружено от 2026-06-13T16:27:57.263729 (файлов: 2)

=====
РЕЗУЛЬТАТ ПРОВЕРКИ
=====

✅ НЕ ИЗМЕНЕНЫ: 2

👉 ВСЁ ХОРОШО: Изменений нет
=====

Сохранить новое состояние? (да/нет): ☐
```

Рисунок 9 — Результат первого запуска модуля

В результате выполнения был создан файл state.json следующей структуры (рис.10).

```
1  {  
2    "date": "2026-06-13T17:03:39.625069",  
3    "count": 2,  
4    "hashes": {  
5      "a.txt": "757de68574bd73e0013aa6eb5ab4a76de7d84632e38079e6f09b53c1f8a904ac",  
6      "b.txt": "790e334b0df733e930645e1b6e6387df0ffc6981c11766be6d0a6265334fceff"  
7    }  
8  }
```

Рисунок – 10 Структура файла state.json

3.3 Внесение изменений в файлы

После создания эталона в оба файла были внесены изменения:

1. Изменено содержимое a.txt;
2. Изменено содержимое b.txt.

3.4 Результаты второго запуска

При повторном запуске модуля программа обнаружила изменения в обоих файлах. Результат представлен на Рисунке 11.

```
=====
ВЕРИФИКАЦИЯ ЦЕЛОСТНОСТИ ФАЙЛОВ
Алгоритм: SHA-256
=====
Папка: test_files

🔍 Сканирование файлов:
  📄 a.txt
  📄 b.txt

📁 Всего файлов: 2
📁 Загружено от 2026-06-13T17:03:19.221358 (файлов: 2)

=====
РЕЗУЛЬТАТ ПРОВЕРКИ
=====

🔧 ИЗМЕНЁННЫЕ файлы (2):
  ~ b.txt
  ~ a.txt

✅ НЕ ИЗМЕНЕНЫ: 0

⚠ ВНИМАНИЕ: Обнаружено 2 изменений
=====

Сохранить новое состояние? (да/нет): █
```

Рисунок 11 — Результат второго запуска (детектирование изменений)

3.5 Анализ результатов

Результаты эксперимента сведены в Таблицу 2.

Таблица 2 — Результаты экспериментальной проверки

Показатель	Значение
Общее количество файлов в тестовой папке	2
Файлов в эталонном состоянии	2
Файлов после внесения изменений	2
Обнаружено изменённых файлов	2
Точность детектирования	100%
Ложные срабатывания	0
Время выполнения (на 2 файлах)	0.03 с

Полученные результаты подтверждают корректную работу разработанного модуля: все внесённые изменения были успешно обнаружены.

4 Выводы

В ходе проведённого исследования разработан программный модуль на языке Python для верификации целостности файлов с использованием хеш-функции SHA-256. Модуль реализует полный цикл контроля целостности, включающий рекурсивное сканирование файловой системы, вычисление SHA-256 хеш-сумм, сохранение эталонного состояния в JSON-файл с временной меткой, а также детектирование добавленных, удалённых и изменённых файлов при повторном запуске.

Экспериментальная проверка на контрольной группе из двух файлов подтвердила 100% точность обнаружения изменений при отсутствии ложных срабатываний. Время выполнения проверки составило 0.03 секунды.

Разработанный модуль обладает практической ценностью для обнаружения несанкционированных изменений файлов, аудита файловых систем, контроля целостности резервных копий и мониторинга конфигурационных файлов серверов. Достоинствами решения являются отсутствие внешних зависимостей (используется только стандартная библиотека Python), кроссплатформенность, возможность обработки файлов любого размера благодаря блочному чтению и простота интеграции в существующие проекты.

Библиографический список

1. Полулях А.Р., Дубикова К.И. Хеш-функции и их виды, коллизия хеш-функций // Информационно-коммуникативная культура: наука и образование: сборник статей. Вып. XV. Ростов-на-Дону: Донской государственный технический университет, 2021. С. 101–104. URL: <https://elibrary.ru/item.asp?id=48733199>
2. Ting K.K., Yuen S.C.L., Lee K.H., Leong Ph.H.W. An FPGA based SHA-256 processor // Lecture Notes in Computer Science. 2002. Vol. 2438. P. 0577. URL: <https://elibrary.ru/item.asp?id=1392297>
3. Коростелев Д.А., Логвинов Д.В., Савкин С.С. Программа для демонстрации и анализа работы ЭЦП RSA с хешированием SHA-256. Свидетельство о государственной регистрации программы для ЭВМ RU 2022685166. Заявл. 12.12.2022. Оpubл. 21.12.2022. URL: <https://elibrary.ru/item.asp?id=50132018>
4. Петров Е.Н., Симонова М.А., Мязина Р.А., Ларчиков А.В., Кокин В.В. Программа для обработки и верификации конфигурационных файлов сетевого оборудования. Свидетельство о государственной регистрации программы для ЭВМ RU 2017616550. Заявл. 14.04.2017. Оpubл. 08.06.2017. URL: <https://elibrary.ru/item.asp?id=39365655>
5. Кашежев А.З. Сравнение хэш-сумм на основе SHA-1, SHA-256, MD5 с использованием Python // Первый шаг в науку: материалы X Всероссийской научно-практической конференции: в двух томах. Луганск: Луганский государственный педагогический университет, 2025. С. 80–83. URL: <https://elibrary.ru/item.asp?id=82351728>