ЛЕКЦИЯ 13

Алгоритмы SHA

Криптографический дайджест помогает сверять данные при наличии доверенного эталона.



Цель и содержание лекции

ЦЕЛЬ И РЕЗУЛЬТАТЫ ОБУЧЕНИЯ

объяснить свойства криптографических хеш-функций, проследить подготовку сообщения для SHA-256 и вычислить дайджест файла целиком и по частям.

- Различать хеширование и шифрование.
- Считать SHA-256 для байтов и для файла.
- Разграничивать проверку целостности, подлинность и хранение паролей.

ПЛАН ЛЕКЦИИ

1. **Дайджест, прообраз и коллизия**
2. **Семейства SHA-1, SHA-2 и SHA-3**
3. **Дополнение сообщения и обработка блоков SHA-256**
4. **Потоковое вычисление в Python и проверка целостности**

Проверка файла по доверенному дайджесту

СХЕМА

1

Эталон

Получаем дайджест из доверенного источника

2

Байты

Читаем локальный файл частями

3

SHA-256

Считаем свой 256-битный дайджест

4

Сверка

Сравниваем вычисленное значение с эталоном

Несовпадение показывает, что проверка не пройдена. Сам по себе хеш не удостоверяет автора.

Что вычисляет криптографическая хеш-функция

01 ТЕМА

Хеш-функция получает байты сообщения и возвращает короткое представление фиксированного размера — дайджест. Для SHA-256 выход состоит из 256 битов, то есть 32 байтов или 64 шестнадцатеричных символов. Одинаковые байты дают одинаковый дайджест при одинаковом алгоритме. Даже небольшое изменение сообщения обычно меняет многие биты дайджеста. По расстоянию между хешами SHA нельзя судить о сходстве двух текстов.

Хеширование используют в хеш-таблицах, но требования там отличаются. Хеш-таблице важно быстро получить индекс и распределить ключи по корзинам. Совпавшие индексы разрешаются, например, сравнением самих ключей в одной корзине. Криптографической функции нужны дополнительные свойства, поскольку злоумышленник может намеренно подбирать входы. Рассмотрим три разных задачи подбора.

Что вычисляет криптографическая хеш-функция

01 ПРОДОЛЖЕНИЕ

Свойство	Что пытаются найти
Стойкость к поиску прообраза	Сообщение с заданным дайджестом
Стойкость к поиску второго прообраза	Другое сообщение с дайджестом заданного сообщения
Стойкость к коллизиям	Любые два различных сообщения с одинаковым дайджестом

Коллизии неизбежны, поскольку множество возможных сообщений больше множества из `2**256` результатов SHA-256. Требование стойкости означает вычислительную трудность поиска. Для идеализированной функции с `b` битами выхода перебор до первого случайного совпадения двух дайджестов занимает порядка `2**(b/2)` вычислений. При подборе под заранее заданный дайджест общий перебор занимает порядка `2**b` попыток, если отсутствует информация о возможных входах. Например, хеш четырехзначного кода можно проверить перебором всех `10 000` кодов. Здесь набор вероятных сообщений мал независимо от длины дайджеста.

Какие алгоритмы называют SHA

02 ТЕМА

SHA — сокращение от Secure Hash Algorithm. SHA-1 дает `160` битов. Для SHA-1 показаны практические коллизии, поэтому для новых задач, где важна стойкость к коллизиям, выбирают алгоритмы семейств SHA-2 или SHA-3. SHA-256 и SHA-512 входят в SHA-2. Их длины дайджестов равны `256` и `512` битам соответственно. SHA3-256 входит в SHA-3, который основан на конструкции Keccak. Его выход той же длины, что у SHA-256, но вычисляется по другой схеме. Обмен дайджестами требует согласования названия алгоритма, входных байтов и формата представления результата.

В Python `hashlib.sha256(data).digest()` возвращает байты, а `hexdigest()` — строку с шестнадцатеричной записью этих же байтов. Для строки сначала выбирают кодировку. Например, символ `я` в UTF-8 занимает два байта, а в Windows-1251 — один. При сравнении файлов следует читать их в двоичном режиме. Это сохраняет байты конца строки и другие данные в исходном виде.

Пошаговый пример SHA-256

Стандарт SHA-256 обрабатывает блоки по `512` битов, или по `64` байта. Подготовка нужна, чтобы разбить данные на блоки и однозначно зафиксировать длину исходного сообщения. К последовательности из целых байтов сначала добавляется `0x80` с единицей в старшем бите и нулями в остальных. Затем добавляются нулевые байты, чтобы длина перед последним полем была равна `56` по модулю `64`. Последнее поле из восьми байтов содержит исходную длину в битах в порядке от старшего байта к младшему.

Для `b"abc"` исходные байты в шестнадцатеричной записи — `61 62 63`, а длина — `24` бита. После `80` добавляются `52` нулевых байта. Последнее поле равно `00 00 00 00 00 00 00 18`, поскольку `0x18 = 24`. Общий размер составляет `3 + 1 + 52 + 8 = 64` байта. Сообщение из `55` байтов еще помещается вместе с дополнением в один блок. При длине `56` байтов нужны два блока.

Пошаговый пример SHA-256

03 ПРОДОЛЖЕНИЕ

У алгоритма есть внутреннее состояние из восьми 32-битных слов. Каждый блок читается как 16 слов и разворачивается в расписание из 64 слов. В 64 раундах рабочие слова смешиваются с расписанием и константами. Среди операций используются циклические сдвиги, побитовое исключающее ИЛИ и сложение по модулю 2**32. После раундов результат прибавляется к текущему состоянию по тому же модулю. Следующий блок начинает работу с обновленного состояния. После последнего блока восемь слов состояния дают дайджест. Блоки обрабатываются в порядке следования в сообщении.

Пошаговый пример SHA-256

13_01.py / фрагмент 1 из 2

```
from hashlib import sha256, sha3_256
from io import BytesIO


def sha256_stream(stream, chunk_size=65536):
    if chunk_size <= 0:
        raise ValueError("Размер порции должен быть положительным")
    result = sha256()
    while True:
        chunk = stream.read(chunk_size)
        if not chunk:
            break
        result.update(chunk)
    return result.hexdigest()
```

Пошаговый пример SHA-256

03 ПРИМЕР PYTHON

13_01.py / фрагмент 2 из 2

```
original = b"abc"
modified = b"abd"
expected = (
    "ba7816bf8f01cfea414140de5dae2223"
    "b00361a396177a9cb410ff61f20015ad"
)
actual = sha256_stream(BytesIO(original), chunk_size=2)
assert actual == sha256(original).hexdigest()
print(actual)
print("Эталон совпал:", actual == expected)
print("После изменения совпал:", sha256(modified).hexdigest() == actual)
print("Байтов в SHA3-256:", len(sha3_256(original).digest()))
```

РЕЗУЛЬТАТ ЗАПУСКА

```
ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad
Эталон совпал: True
После изменения совпал: False
Байтов в SHA3-256: 32
```

Пошаговый пример SHA-256

03 ПРОДОЛЖЕНИЕ

`BytesIO` имитирует двоичный файл. Для файла на диске передают поток из `open(path, "rb")`. Вызовы `update(b"ab")` и `update(b"c")` подают вместе ту же последовательность байтов, что один вызов `update(b"abc")`. Порция чтения программы и блок SHA-256 — разные уровни обработки. Библиотека буферизует неполные блоки и добавляет длину при вычислении результата. Для файла из `n` байтов затраты времени составляют `O(n)`, а дополнительная память — `O(s)` при размере порции `s`.

Что можно заключить из совпадения дайджестов

Пусть издатель помещает рядом с названием файла ожидаемый SHA-256. Получатель вычисляет дайджест скачанных байтов и сравнивает его с ожидаемым. Если значения различны, содержимое отличается от эталона либо выбраны разные параметры вычисления. При совпадении и доверенном источнике эталона случайная порча крайне маловероятна. Если файл и опубликованный дайджест подменены одновременно, обычное сравнение не обнаружит такую подмену. Для подтверждения источника используют цифровую подпись или, при общем секретном ключе, код аутентификации сообщения HMAC.

Хеширование паролей предъявляет другое требование. Если хранилище утечет, проверка каждой догадки должна требовать заметных ресурсов. Обычный SHA-256 специально устроен для быстрого вычисления и позволяет быстро проверять догадки. Для хранения паролей используют специально предназначенную функцию с индивидуальной случайной солью и настраиваемой стоимостью, например Argon2id. Соль отличает вычисления для двух одинаковых паролей у разных пользователей. Стоимость функции повышает затраты массового перебора.

Что нужно уметь после лекции

ИТОГИ

- SHA преобразует сообщение в дайджест фиксированной длины. Длина и способ вычисления зависят от конкретного алгоритма.
- SHA-256 дополняет сообщение, обрабатывает его блоками и обновляет внутреннее состояние.
- Потоковое вычисление дает тот же результат, что обработка тех же байтов одним вызовом.
- Вывод о целостности зависит от доверия к эталону. Аутентификация требует подписи или конструкции с ключом.

> Криптографический дайджест помогает сверять данные при наличии доверенного эталона.

Проверьте понимание

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Почему коллизии существуют даже у стойкой хеш-функции?
2. Чем поиск второго прообраза отличается от поиска произвольной коллизии?
3. Для чего SHA-256 добавляет исходную длину сообщения?
4. Когда совпадение дайджеста скачанного файла с опубликованным эталоном дает основание доверять содержимому?

Самостоятельная работа

ПРАКТИКА

1. Найдите число байтов дополнения для сообщений длиной `0`, `55`, `56` и `64` байта, учитывая поле длины.
2. Вычислите SHA-256 строк `"данные"` и `"данные\n"` в UTF-8. Сравните дайджесты и длины входных последовательностей.
3. Для одного файла сравните дайджесты при размерах порции `1`, `64` и `4096` байтов. Подтвердите совпадение с помощью `assert`.

Источники и документация

ЧТЕНИЕ

ОСНОВНОЕ ЧТЕНИЕ

А. Бхаргава. Грокаем алгоритмы

СПб.: Питер, 2017. Глава 11. «Алгоритмы SHA». С. 266–269.

ДОПОЛНИТЕЛЬНО

NIST. FIPS 180-4, Secure Hash Standard. https://csrc.nist.gov/pubs/fips/180-4/upd1/final

NIST. FIPS 202, SHA-3 Standard. https://csrc.nist.gov/pubs/fips/202/final

Python. Документация модуля `hashlib`. https://docs.python.org/3/library/hashlib.html

RFC 9106. Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications. https://www.rfc-editor.org/rfc/rfc9106