15. Функции хеширования и целостность данных. Криптографические функции хеширования. Хеш-функции на основе симметричных блочных алгоритмов

Что такое хеш и хэширование простыми словами

Слово **хеш** происходит от английского «hash», одно из значений которого трактуется как путаница или мешанина. Собственно, это довольно полно описывает реальное значение этого термина. Часто еще про такой процесс говорят «хеширование», что опять же является производным от английского hashing (рубить, крошить, спутывать...).

Появился этот термин в середине прошлого века среди людей занимающихся обработках массивов данных. Хеш-функция позволяла привести любой массив данных к числу заданной длины. Например, если любое число (любой длинны) начать делить много раз подряд на одно и то же простое число, то полученный в результате остаток от деления можно будет называть хешем. Для разных исходных чисел остаток от деления (цифры после запятой) будет отличаться.

Для обычного человека это кажется белибердой, но как ни странно в наше время без хеширования практически невозможна работа в интернете. Так что же это такая за функция? На самом деле она может быть любой (приведенный выше пример это не есть реальная функция — он придуман мною чисто для вашего лучшего понимания принципа). Главное, чтобы результаты ее работы удовлетворяли приведенным ниже условиям.

Зачем нужен хэш

Смотрите, еще пример. Есть у вас текст в файле. Но на самом деле это ведь не текст, а массив цифровых символов (по сути число). Как вы знаете, в компьютерной логике используются двоичные числа (ноль и единица). Они запросто могут быть преобразованы в шестнадцатиричные цифры, над которыми можно проводить математические операции. Применив к ним хеш-функцию мы получим на выходе (после ряда итераций) число заданной длины (хеш-сумму).

Если мы потом в исходном текстовом файле поменяем хотя бы одну букву или добавим лишний пробел, то повторно рассчитанный для него хэш уже будет отличаться от изначального (вообще другое число будет), зачем все это нужно? Ну, конечно же, для того, чтобы понять, что файл именно тот, что и должен быть. Это можно использовать в целом ряде аспектов работы в интернете и без этого вообще сложно представить себе работу сети.

Где и как используют хеширование

Например, простые хэш-функции (не надежные, но быстро рассчитываемые) применяются при проверке целостности передачи пакетов по протоколу TCP/IP (и ряду других протоколов и алгоритмов, для выявления аппаратных ошибок и сбоев — так называемое избыточное кодирование). Если рассчитанное значение хеша совпадает с отправленным вместе с пакетом (так называемой контрольной суммой), то значит потерь по пути не было (можно переходить к следующему пакету).

А это, ведь на минутку, основной протокол передачи данных в сети интернет. Без него никуда. Да, есть вероятность, что произойдет накладка — их называют коллизиями. Ведь для разных изначальных данных может получиться один и тот же хеш. Чем проще используется функция, тем выше такая вероятность. Но тут нужно просто выбирать между тем, что важнее в данный момент — надежность идентификации или скорость работы. В случае TCP/IP важна именно скорость. Но есть и другие области, где важнее именно надежность.

Похожая схема используется и в технологии блокчейн, где хеш выступает гарантией целостности цепочки транзакций (платежей) и защищает ее от несанкционированных изменений. Благодаря ему и распределенным вычислениям взломать блокчен очень сложно и на его основе благополучно существует множество криптовалют, включая самую популярную из них — это биткоин. Последний существует уже с 2009 год и до сих пор не был взломан.

Более сложные хеш-функции используются в криптографии.

Главное условие для них — невозможность по конечному результату (хэшу) вычислить начальный (массив данных, который обработали данной хеш-функцией).

Второе главное условие — стойкость к коллизиями, т.е. низкая вероятность получения двух одинаковых хеш-сумм из двух разных массивов данных при обработке их этой функцией. Расчеты по таким алгоритмам более сложные, но тут уже главное не скорость, а надежность.

Так же хеширование используется в технологии электронной цифровой подписи. С помощью хэша тут опять же удостоверяются, что подписывают именно тот документ, что требуется. Именно он (хеш) передается в токен, который и формирует электронную цифровую подпись.

Для доступа к сайтам и серверам по логину и паролю тоже часто используют хеширование. Согласитесь, что хранить пароли в открытом виде (для их сверки с вводимыми пользователями) довольно ненадежно (могут их похитить).

Поэтому хранят хеши всех паролей. Пользователь вводит символы своего пароля, мгновенно рассчитывается его хеш-сумма и сверяется с тем, что есть в базе. Надежно и очень просто. Обычно для такого типа хеширования используют сложные функции с очень высокой криптостойкостью, чтобы по хэшу нельзя было бы восстановить пароль.

Коллизии хэш-функций

В теории хэш-функций предусмотрено такое явление, как коллизия. В чем его сущность? Коллизия хэш-функции - ситуация, при которой два разных файла имеют одинаковый хэш-код. Это возможно, если длина целевой последовательности символов будет небольшой. В этом случае вероятность совпадения хэша будет выше.

Для того чтобы избежать коллизии, рекомендуется, в частности, задействовать двойной алгоритм под названием "хеширование хеш-функции". Он предполагает формирование открытого и закрытого кода.

Многие программисты при решении ответственных задач рекомендуют не применять хэш-функции в тех случаях, когда это необязательно и всегда тестировать соответствующие алгоритмы на предмет наилучшей совместимости с теми или иными ключами.

Какими свойствами должна обладать хеш-функция:

Хочу систематизировать кое-что из уже сказанного и добавить новое.

1. Как уже было сказано, функция эта должна уметь приводить любой объем данных (а все они цифровые, т.е. двоичные, как вы понимаете) к числу заданной длины (по сути это сжатие до битовой последовательности заданной длины хитрым способом).
2. При этом малейшее изменение (хоть на один бит) входных данных должно приводить к полному изменению хеша.
3. Она должна быть стойкой в обратной операции, т.е. вероятность восстановления исходных данных по хешу должна быть весьма низкой (хотя последнее сильно зависит от задействованных мощностей)
4. В идеале она должна иметь как можно более низкую вероятность возникновения коллизий. Согласитесь, что не айс будет, если из разных массивов данных будут часто получаться одни и те же значения хэша.
5. Хорошая хеш-функция не должна сильно нагружать железо при своем исполнении. От этого сильно зависит скорость работы системы на ней построенной. Как я уже говорил выше, всегда имеется компромисс между скорость работы и качеством получаемого результата.
6. Алгоритм работы функции должен быть открытым, чтобы любой желающий мог бы оценить ее криптостойкость, т.е. вероятность восстановления начальных данных по выдаваемому хешу.

Хеш — это маркер целостности скачанных в сети файлов

Где еще можно встретить применение этой технологии? Наверняка при скачивании файлов из интернета вы сталкивались с тем, что там приводят некоторые числа (которые называют либо хешем, либо контрольными суммами) типа:

CRC32: 7438E546
MD5: DE3BAC46D80E77ADCE8E379F682332EB
SHA-1: 332B317FB97126B0F79F7AF5786EBC51E5CC82CF

Что это такое? И что вам с этим всем делать? Ну, как правило, на тех же сайтах можно найти пояснения по этому поводу, но я не буду вас утруждать и расскажу в двух словах. Это как раз и есть результаты работы различных хеш-функций (их названия приведены перед числами: CRC32, MD5 и SHA-1).

Зачем они вам нужны? Ну, если вам важно знать, что при скачивании все прошло нормально и ваша копия полностью соответствует оригиналу, то нужно будет поставить на свой компьютер программку, которая умеет вычислять хэш по этим алгоритмам (или хотя бы по некоторым их них).

После чего прогнать скачанные файлы через эту программку и сравнить полученные числа с приведенными на сайте. Если совпадают, то сбоев при скачивании не было, а если нет, то значит были сбои и есть смысл повторить закачку заново.

Популярные хэш-алгоритмы сжатия

1. **CRC32** — используется именно для создания контрольных сумм (так называемое избыточное кодирование). Данная функция не является криптографической. Есть много вариаций этого алгоритма (число после CRC означает длину получаемого хеша в битах), в зависимости от нужной длины получаемого хеша. Функция очень простая и нересурсоемкая. В связи с этим используется для проверки целостности пакетов в различных протоколах передачи данных.
2. **MD5** — старая, но до сих пор очень популярная версия уже криптографического алгоритма, которая создает хеш длиной в 128 бит. Хотя стойкость этой версии на сегодняшний день и не очень высока, она все равно часто используется как еще один вариант контрольной суммы, например, при скачивании файлов из сети.
3. **SHA-1** — криптографическая функция, формирующая хеш-суммы длиной в 160 байт. Сейчас идет активная миграция в сторону SHA-2, которая обладает более высокой устойчивостью, но SHA-1 по-прежнему активно используется хотя бы в качестве контрольных сумм. Но она так же по-прежнему используется и для хранения хешей паролей в базе данных сайта (об этом читайте выше).
4. **ГОСТ Р 34.11-2012** — текущий российский криптографический (стойкий к взлому) алгоритм введенный в работу в 2013 году (ранее использовался ГОСТ Р 34.11-94). Длина выходного хеша может быть 256 или 512 бит. Обладает высокой криптостойкостью и довольно хорошей скоростью работы. Используется для электронных цифровых подписей в системе государственного и другого документооборота.

Требования к хэш-функциям

Существует ряд требований к хэш-функциям, предназначенным для практического задействования в той или иной области. Во-первых, соответствующий алгоритм должен характеризоваться чувствительностью к изменениям во внутренней структуре хешируемых документов. То есть в хэш-функции должны распознаваться, если речь идет о текстовом файле, перестановки абзацев, переносы.

С одной стороны, содержимое документа не меняется, с другой — корректируется его структура, и этот процесс должен распознаваться в ходе хеширования. Во-вторых, рассматриваемый алгоритм должен преобразовывать данные так, чтобы обратная операция (превращение хэша в изначальный документ) была на практике невозможна. В-третьих, хэш-функция должна предполагать задействование таких алгоритмов, которые практически исключают вероятность формирования одинаковой последовательности символов в виде хэш, иными словами — появления так называемых коллизий.

Их сущность мы рассмотрим чуть позже. Отмеченные требования, которым должен соответствовать алгоритм хэш-функции, могут быть обеспечены главным образом за счет задействования сложных математических подходов.

Структура

Изучим то, какой может быть структура рассматриваемых функций. Как мы отметили выше, в числе главных требований к рассматриваемым алгоритмам — обеспечение одно направленности шифрования. Человек, имеющий в распоряжении только хэш, практически не должен иметь возможности получить на его основе исходный документ.

В какой структуре может быть представлена используемая в подобных целях хеш-функция? Пример ее составления может быть таким: H (hash, то есть, хэш) = f (T (текст), H1), где H1 — алгоритм обработки текста T. Данная функция хеширует T таким образом, что без знания H1 открыть его как полноценный файл будет практически невозможно.

Использование хэш-функций на практике: скачивание файлов

Изучим теперь подробнее варианты использования хэш-функций на практике. Задействование соответствующих алгоритмов может применяться при написании скриптов скачивания файлов с интернет-серверов. В большинстве случаев для каждого файла определяется некая контрольная сумма — это и есть хэш. Она должна быть одинаковой для объекта, располагающегося на сервере и скачанного на компьютер пользователя. Если это не так, то файл может не открыться либо запуститься не вполне корректно.

Хэш-функция и ЭЦП

Использование хэш-функций распространено при организации обмена документами, содержащими электронно-цифровую подпись. Хэшируется в данном случае подписываемый файл, для того чтобы его получатель мог удостовериться в том, что он подлинный. Хотя формально хэш-функция не входит в структуру электронного ключа, она может фиксироваться во флеш-памяти аппаратных средств, с помощью которых подписываются документы, таких как, например, eToken.

Электронная подпись представляет собой шифрование файла при задействовании открытого и закрытого ключей. То есть к исходному файлу прикрепляется зашифрованное с помощью закрытого ключа сообщение, а проверка ЭЦП осуществляется посредством открытого ключа. Если хэш-функция обоих документов совпадает — файл, находящийся у получателя, признается подлинным, а подпись отправителя распознается как верная.

Хеширование, как мы отметили выше, не является непосредственно компонентом ЭЦП, однако позволяет весьма эффективно оптимизировать алгоритмы задействования электронной подписи.

Так, шифроваться может, собственно, только хэш, а не сам документ. В итоге скорость обработки файлов значительно возрастает, одновременно становится возможным обеспечивать более эффективные механизмы защиты ЭЦП, так как акцент в вычислительных операциях в этом случае будет ставиться не на обработке исходных данных, а на обеспечении криптографической стойкости подписи. Хэш-функция к тому же делает возможным подписывать самые разные типы данных, а не только текстовые.

Проверка паролей

Еще одна возможная область применения хеширования — организация алгоритмов проверки паролей, установленных для разграничения доступа к тем или иным файловым ресурсам. Каким образом при решении подобных задач могут быть задействованы те или иные виды хеш-функций? Очень просто.

Дело в том, что на большинстве серверов, доступ к которым подлежит разграничению, пароли хранятся в виде хэшированных значений. Это вполне логично — если бы пароли были представлены в исходном текстовом виде, хакеры, получившие доступ к ним, могли бы запросто читать секретные данные. В свою очередь, на основе хэш вычислить пароль непросто.

Каким образом осуществляется проверка доступа пользователя при задействовании рассматриваемых алгоритмов?

Пароль, вводимый пользователем, сверяется с тем, что зафиксирован в хэш-функции, что хранится на сервере. Если значения текстовых блоков совпадают — пользователь получает необходимый доступ к ресурсам. В качестве инструмента проверки паролей может

быть задействована самая простая хэш-функция. Но на практике IT-специалисты чаще всего используют комплексные многоступенчатые криптографические алгоритмы. Как правило, они дополняются применением стандартов передачи данных по защищенному каналу — так, чтобы хакеры не смогли обнаружить либо вычислить пароль, передаваемый с компьютера пользователя на сервера — до того, как он будет сверяться с хешированными текстовыми блоками.

Однонаправленные хэш-функции на основе симметричных блочных алгоритмов

Однонаправленную хэш-функцию можно построить, используя симметричный блочный алгоритм. Наиболее очевидный подход состоит в том, чтобы шифровать сообщение M посредством блочного алгоритма в режиме CBC или CFB с помощью фиксированного ключа и некоторого вектора инициализации IV.

Последний блок шифртекста можно рассматривать в качестве хэш-значения сообщения M. При таком подходе не всегда возможно построить безопасную однонаправленную хэш-функцию, но всегда можно получить код аутентификации сообщения MAC (Message Authentication Code).

Примечание: Режимы работы алгоритма DES: CBC (Cipher Block Chaining)- сцепление блоков шифра; CFB (Cipher Feed Back) обратная связь по шифртексту)

Более безопасный вариант хэш-функции можно получить, используя:

- блок сообщения в качестве ключа,
- предыдущее хэш-значение - в качестве входа,
- а текущее хэш-значение - в качестве выхода.

Реальные хэш-функции проектируются еще более сложными:

1. длина блока обычно определяется длиной ключа
2. длина хэш-значения совпадает с длиной блока.

Поскольку большинство блочных алгоритмов являются 64-битовыми, некоторые схемы хэшированияпроектируют так, чтобы хэш-значение имело длину, равную двойной длине блока.

Если принять, что получаемая хэш-функция корректна, безопасность схемы хэширования базируется на безопасности лежащего в ее основе блочного алгоритма.
Схема хэширования, у которой длина хэш-значения равна длине блока, показана на рис.3.

Ее работа описывается выражениями:

$H0 = Iн$,

$Hi = EA(B) \oplus C$,

где $\oplus$ - сложение по модулю 2 (исключающее ИЛИ); Iн - некоторое случайное начальное значение; A, B, C могут принимать значения Mi, Hi-1, (Mi $\oplus$ Hi-1) или быть константами.

Рис.3. Обобщенная схема формирования хэш-функции

Сообщение M разбивается на блоки Mi принятой длины, которые обрабатываются поочередно.

Три различные переменные A, B, C могут принимать одно из четырех возможных значений, поэтому в принципе можно получить 64 варианта общей схемы этого типа. Из них 52 варианта являются либо тривиально слабыми, либо небезопасными. Остальные 12 схем безопасного хэширования, у которых длина хэш-значения равна длине блока перечислены в табл.1.

Таблица 1

Номер схемы	Функция хэширования
1	$H_i = E_{Hi-1} (M_i) \oplus M_i$
2	$H_i = E_{Hi-1} (M_i \oplus H_{i-1}) \oplus M_i \oplus H_{i-1}$
3	$H_i = E_{Hi-1} (M_i) \oplus M_i \oplus H_{i-1}$
4	$H_i = E_{Hi-1} (M_i \oplus H_{i-1}) \oplus M_i$
5	$H_i = E_{Mi} (H_{i-1}) \oplus H_{i-1}$
6	$H_i = E_{Mi} (M_i \oplus H_{i-1}) \oplus M_i \oplus H_{i-1}$
7	$H_i = E_{Mi} (H_{i-1}) \oplus M_i \oplus H_{i-1}$
8	$H_i = E_{Mi} (M_i \oplus H_{i-1}) \oplus H_{i-1}$
9	$H_i = E_{Mi \oplus Hi-1} (M_i) \oplus M_i$
10	$H_i = E_{Mi \oplus Hi-1} (H_{i-1}) \oplus H_{i-1}$
11	$H_i = E_{Mi \oplus Hi-1} (M_i) \oplus H_{i-1}$
12	$H_i = E_{Mi \oplus Hi-1} (H_{i-1}) \oplus M_i$

Первые четыре схемы хэширования, являющиеся безопасными при всех атаках, приведены на рис.4.

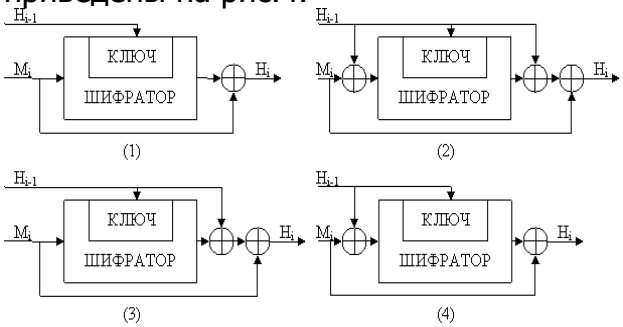

Рис.4. Четыре схемы безопасного хэширования

Недостатком хэш-функций, спроектированных на основе блочных алгоритмов, является несколько заниженная скорость работы.

Дело в том, что ту же самую стойкость относительно двух основных требований к хэш-функции можно обеспечить за гораздо меньшее количество операций над входными данными. Но для этого алгоритм необходимо изначально проектировать специально, исходя из тандема требований (стойкость, скорость).