Часть I

Основы криптографии

Человечество каждый день полагается на хеширование, шифрование и цифровые подписи. Но вся слава обычно достается шифрованию. О нем чаще говорят на выступлениях и лекциях, на него чаще обращают внимание журналисты. Программистам тоже, как правило, любопытнее всего именно оно.

В первой части книги регулярно обращается внимание на то, почему без цифровых подписей и вычисления хешей невозможно обойтись – настолько же, насколько и без шифрования. Последующие части тоже постоянно напоминают об этом. Главы 2–6 полезно прочитать даже в отрыве от всей книги, но они будут хорошим подспорьем в понимании дальнейшего материала.

2 Хеширование

Темы этой главы:

- что такое хеш-функции;
- знакомство с архетипичными персонажами;
- проверка целостности данных с помощью хеширования;
- как выбрать криптографическую хеш-функцию;
- модуль `hashlib` для подсчета криптохешей.

В этой главе рассказывается о том, как применять хеш-функции для проверки целостности данных. Это необходимо для построения любой защищенной системы. Также говорится о том, как различать безопасные и небезопасные хеш-функции. Между делом мы познакомимся с Алисой, Бобом и другими архетипичными персонажами. Их обыкновенно можно увидеть в схемах атаки и защиты приложений. Эта книга не исключение. И в конце следует рассказ о модуле `hashlib`, с помощью которого и будут высчитываться хеши.

2.1 Что такое хеш-функция?

Любой *хеш-функции* можно подать на вход данные и получить на выходе результат. Входные данные хеш-функции называются *сообщением*. Сообщением могут быть любые данные. «Война и мир», кар-

тинка с котиком, пакет Python – все это может служить сообщением. На выходе хеш-функция выдает очень большое число. Это число носит много названий: *хеш*, *хеш-сумма*, *отпечаток*.

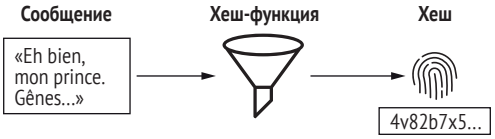

Рис. 2.1 Хеш-функция превращает входные данные, то есть сообщение, в хеш

В этой книге оно будет называться *хеш*. Хеш обычно представляет из себя строку из латинских букв и арабских цифр. Хеш-функция превращает набор любых сообщений в набор хешей. Рисунок 2.1 схематично описывает, как эти понятия взаимодействуют между собой.

В нашей книге для обозначения хеш-функций используется воронка. Что хеш-функция, что воронка принимают на вход содержимое неопределенного размера, но размеры результата предопределены. Хеш обозначается отпечатком пальца. Отпечаток уникален для человека, хеш уникален для сообщения, и оба могут быть использованы для опознания.

Хеш-функции отличаются друг от друга. Различия в общем сводятся к их свойствам, о которых рассказано чуть ниже. Чтобы познакомиться с некоторыми свойствами, нам пригодится встроенная в Python функция с говорящим названием `hash`. Она используется в Python для обработки словарей (dictionary) и наборов данных (set). Нам она пригодится в качестве примера.

Встроенная функция `hash` отлично подходит для знакомства с понятиями, так как она весьма незатейливее других хеш-функций, о которых мы поговорим дальше. Этой функции требуется один аргумент, сообщение, и на выходе получается хеш:

```
$ python
>>> message = 'message'  <-- Сообщение, поданное на вход
>>> hash(message)
2010551929503284934      <-- Хеш на выходе
```

Хеш-функциям присущи три основных свойства:

- детерминированное поведение;
- хеши неизменной длины;
- лавинный эффект.

Детерминированное поведение

Любая хеш-функция является *детерминированной*: для одного и того же сообщения на входе функция всегда выдает одинаковый результат. Иначе говоря, хеш-функциям присуща воспроизводимость

полученного результата, он не случаен. Встроенная функция `hash` всегда возвращает один и тот же хеш для любого отдельно взятого сообщения в рамках одного процесса Python. Запустите следующие две строки кода в интерактивной консоли Python. Полученные вами значения хешей будут совпадать друг с другом, но будут отличаться от полученных автором:

```
>>> hash('same message')
1116605938627321843
>>> hash('same message')
1116605938627321843
```

Хеш-функции, которые обсуждаются далее, абсолютно детерминированные. Они выдают один и тот же результат вне зависимости ни от чего.

Хеши неизменной длины

У сообщений произвольная длина, у хешей же для каждой хеш-функции длина строго определена. Если функции не присуще это качество, то она не может считаться хеш-функцией. Длина сообщения не должна влиять на длину хеша. Подача на вход встроенной функции `hash` различных сообщений даст на выходе различные хеши, но каждое значение всегда будет целым числом в заданных пределах.

Лавинный эффект

Когда незначительное изменение сообщения разительно сказывается на получаемом хеше, это означает, что хеш-функции присущ лавинный эффект. В идеальном случае каждый бит хеша зависит от каждого бита входных данных. Если два сообщения различаются хотя бы на бит, тогда в среднем только половина битов хеша должны совпасть. По каждой конкретной хеш-функции судят отдельно, насколько она близка к идеалу.

Взгляните на данный код. Значения хешей как для строки, так и для целого числа имеют заданную длину, но только на хеши от строк действует лавинный эффект.

```
>>> bin(hash('a'))
'0b1001001101100101101100100011100111100111111011101010000111100010'
>>> bin(hash('b'))
'0b1011110111111110110110010100110000001010000011110100010111001110'
>>>
>>> bin(hash(0))
'0b0'
>>> bin(hash(1))
'0b1'
```

Встроенная функция `hash` подходит для образовательных целей, но ее нельзя назвать криптографической хеш-функцией по трем причинам. Следующий раздел рассказывает о них.

2.1.1 Свойства криптографических хеш-функций

Криптографическая хеш-функция должна обладать тремя дополнительными признаками:

- быть вычислительно необратимой;
- иметь слабое сопротивление поиску коллизий;
- иметь сильное сопротивление поиску коллизий.

По-научному это называется *сопротивление поиску прообраза*, *сопротивление поиску второго прообраза*, *стойкость к коллизиям*. Для целей книги эти термины можно не использовать, никакого умысла задеть специалистов в этом нет.

Вычислительно необратимые функции

Все криптографические хеш-функции без исключения обязаны быть *вычислительно необратимыми* (one-way functions). Эта такая функция, результат которой легко вычисляется, но найти аргумент по результату трудно. Иначе говоря, по выходным данным должно быть сложно определить входные. Если злоумышленнику попал в руки хеш, нам нужно, чтобы определить сообщение для этого хеша было очень непросто.

Насколько непросто? Можно сказать, что *недостижимо*. Это означает *очень сложно* – настолько сложно, что взломщику не остается другого способа, кроме метода «грубой силы».

Что подразумевается под «грубой силой»? Любому злоумышленнику, даже не особо смышленому, под силу написать несложный скрипт, чтобы создать очень много сообщений, вычислить хеш от каждого и сравнить этот хеш с имеющимся. Для этого атакующему нужно много времени и вычислительных мощностей: сила есть – ума не надо. Иногда этот метод еще называется *полным перебором*.

Сколько же потребуется времени и мощностей? Сложно сказать, это величина переменная. Если говорить о хеш-функциях, которые будут обсуждаться в дальнейшем, то потребуются миллиарды долларов и миллионы лет. Это то, что специалист по безопасности счел бы недостижимым. Но недостижимым не значит невозможным. Стоит признать, что идеальных хеш-функций не существует, так как все они подвержены перебору «грубой силой».

Недостижимое вполне может стать достижимым. То, что невозможно было подобрать полным перебором лет тридцать назад, может быть успешно подобрано сегодня или в недалеком будущем. Компьютерные комплектующие продолжают дешеветь, вместе с ними дешевеет и применение атаки перебором. Увы, шифры со временем теряют былую стойкость. Это не значит, что любая система рано или поздно будет подвержена взлому. Это значит, что любой системе со временем следует переходить на более стойкие хеш-функции. В этой главе еще будет рассказано, как разумно выбирать хеш-функцию.

Сопротивление поиску коллизий

Все без исключения используемые в криптографии хеш-функции должны обладать *сопротивлением к поиску коллизий*. Что такое коллизия? Несмотря на то что длина хешей разных сообщений одинакова, их значение всегда является различным. Почти всегда. Когда хеш двух разных сообщений оказывается одинаковым, это и называется коллизией. Коллизия – это плохо. Хеш-функции проектируются так, чтобы свести их возникновение к минимуму. Стойкость функции к появлению коллизий – это важный фактор. Некоторые функции справляются лучше, некоторые – хуже.

Если для заданного сообщения найти другое сообщение с таким же хешем недостижимо, то такая хеш-функция обладает *слабым сопротивлением поиску коллизий*. Другими словами, если у взломщика на руках есть входные данные, найти другие данные с таким же хешем ему должно быть недостижимо.

Если недостижимо обнаружить какую бы то ни было коллизию в принципе, то такая хеш-функция обладает *сильным сопротивлением поиску коллизий*. Разница между сильным и слабым сопротивлениями едва заметна. Слабое сопротивление касается нахождения коллизии хеша от известного, предварительно заданного сообщения. Сильное сопротивление касается нахождения коллизий хеша между любыми двумя сообщениями. На рис. 2.2 разница отображена наглядно.

Рис. 2.2 Слабое и сильное сопротивления поиску коллизий

Если функция обладает сильным сопротивлением поиску коллизий, то обладает и слабым. Обратное же неверно. Хеш-функции со слабым сопротивлением поиску коллизий необязательно присуще

сильное сопротивление. Таким образом, реализация сильного сопротивления – весьма непростая задача. Если взломщик либо исследователь находит брешь в криптографической хеш-функции, то именно это свойство она теряет первым. Чуть позже вы увидите живые примеры таких атак.

Вся соль – в слове *недостижимо*. И хотелось бы повстречать хеш-функцию без коллизий, но такой просто-напросто не существует. Смотрите сами: сообщения могут быть любой длины, хеши могут быть только одной. Количество возможных сообщений заведомо превышает количество возможных хешей. *Принцип Дирихлé* в действии.

В этом разделе было рассказано о том, что такое хеш-функция. Теперь перейдем к тому, как с помощью хеширования проверяется целостность данных. Но сперва позвольте познакомить вас с горсткой архетипичных персонажей. В течение книги они часто будут играть роль в пояснительных схемах, начиная как раз с понятия целостности данных.

2.2 Архетипичные персонажи

В схемах и иллюстрациях этой книги можно повстречать пятерых архетипичных персонажей. Вы можете увидеть их на рис. 2.3. Гарантирую вам, с их помощью вам будет куда проще понять материал, а мне – объяснить его. Задачи в этой книге построены вокруг ситуаций, в которых оказываются Алиса и Боб. Если вы читали другие книги о безопасности, то они вам уже должны быть знакомы. Алиса и Боб, как и вы, хотят безопасно создавать данные и обмениваться ими. Иногда к ним будет присоединяться их друг Чарли. Данные в целом будут пересылаться между ними. Алиса, Боб и Чарли – положительные персонажи. Можете представлять себя на их месте.

Рис 2.3 Положительные архетипичные персонажи обозначены нимбом, взломщикам пририсованы рожки

Ева и Мэллори – отрицательные действующие лица. Они атакуют Алису и Боба: пытаются похитить либо подменить данные между ними, а также пытаются выдать себя за них. Ева – пассивный обозрева*т*ель. Из всего пространства для атаки она скорее выберет обмен данными по сети. Мэллори – активная взломщица, она атакует куда

более изощренно. Чаще всего точкой проникновения она выбирает саму систему либо пользователей сервиса.

Запомните их, они вам еще встретятся. У Алисы, Боба и Чарли светятся нимбы. У Евы и Мэллори растут рожки. В следующем разделе Алиса воспользуется хешированием для проверки целостности данных.

2.3 Целостность данных

Целостность данных, иногда *целостность сообщения*, позволяет быть уверенным, что данные не были непреднамеренно изменены. Она дает ответ на вопрос «изменены ли данные?». Допустим, Алиса хранит деловые бумаги в системе документооборота. Сейчас, чтобы отслеживать целостность данных, в системе хранится две копии каждого документа. Чтобы проверить их целостность, они сравниваются побайтово. Если копии разнятся, документ считается искаженным. Алиса недовольна тем, сколько места для хранения потребляет система. Этот сервис уже обходится в копеечку, и чем больше в нем документов, тем дороже стоит его содержать.

Алиса понимает, что стоит перед распространенной проблемой, и находит для нее распространенное решение. Она решает применить криптографическую хеш-функцию. При создании документа система высчитывает и сохраняет его хеш. Чтобы удостовериться в целостности данных, приложение заново высчитывает хеш и сравнивает его с сохраненным ранее значением. Если значения хешей не совпадают, документ считается искаженным.

Рисунок 2.4 пошагово описывает процесс. Фрагмент пазла означает процесс сравнения двух хешей.

Рис. 2.4 Алиса убеждается в целостности данных, сравнивая хеши, а не документы

На этом примере ясно, почему сопротивление поиску коллизий – это важное свойство криптографической хеш-функции. Допустим,

Алиса бы использовала хеш-функцию, допускающую коллизии. Сервис бы просто не смог определить искаженный файл, если бы хеш оригинального и искаженного файла оказался одинаковым.

Этот раздел продемонстрировал важную область применения хеширования для определения целостности данных. В следующем разделе рассказывается о том, как из существующих хеш-функций выбрать подходящую для этой задачи.

2.4 Выбор криптографической хеш-функции

В Python уже встроена поддержка криптографического хеширования. Сторонние фреймворки либо библиотеки для этого не понадобятся. Встроенный модуль `hashlib` предлагает все, что может понадобиться большинству разработчиков для криптографического хеширования. В множестве `algorithms_guaranteed` хранятся все хеш-функции, которые гарантированно поддерживаются на всех платформах. Из этого множества вам и предстоит выбирать. Мало кому требуются функции за пределами данного набора:

```
>>> import hashlib
>>> sorted(hashlib.algorithms_guaranteed)
['blake2b', 'blake2s', 'md5', 'sha1', 'sha224', 'sha256', 'sha384',
'sha3_224', 'sha3_256', 'sha3_384', 'sha3_512', 'sha512', 'shake_128',
'shake_256']
```

Вряд ли вам хоть когда-то повстречаются хеш-функции не из этого списка.

Само собой, настолько широкий выбор может озадачить. И прежде чем выбирать, нужно разделить эти функции на безопасные и небезопасные.

2.4.1 Безопасные хеш-функции

Безопасные хеш-функции из списка `algorithms_guaranteed` принадлежат семействам:

- SHA-2;
- SHA-3;
- BLAKE2.

SHA-2

Семейство хеш-функций SHA-2 было представлено NSA в 2001 году. Оно состоит из функций SHA-224, SHA-256, SHA-384 и SHA-512. Основными функциями являются SHA-256 и SHA-512. Можете не запоминать их названия, пока что нас интересует только SHA-256. Она вам еще не раз встретится на протяжении книги.

Для криптографического хеширования по умолчанию стоит использовать SHA-256. Это очевидный выбор, ведь эта функция уже применяется в любом сервисе. Операционные системы и сетевые протоколы, поверх которых работает приложение, уже полагаются на SHA-256. Выбирать не приходится: пришлось бы серьезно постараться никак не задействовать эту функцию. Она является безопасной, широко поддерживается и используется повсеместно.

В названиях всех функций SHA-2 уже указана длина их хешей. О хеш-функции часто судят по длине ее хеша, и его длина нередко фигурирует в названии. SHA-256, например, выдает хеш длиной, как вы уже догадались, 256 бит. Чем длиннее хеш, тем вероятнее, что он уникален, и тем меньше вероятность коллизии. Чем длиннее, тем лучше.

SHA-3

Семейство хеш-функций SHA-3 состоит из SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128 и SHAKE256. Семейство SHA-3 безопасно, и оно считается наследником SHA-2. Увы, на момент написания книги оно еще не набрало популярности. Стоит подумать об использовании функции этого семейства, например SHA3-256, если требуется повышенная безопасность. Но не забывайте, что поддержка данного семейства не настолько широкая, как у SHA-2.

BLAKE2

Алгоритм BLAKE2 не настолько популярен, насколько SHA-2 или SHA-3, но у него есть козырь в рукаве. BLAKE2 умело использует возможности современных ЦП, чтобы считать хеши на сверхвысоких скоростях. Именно поэтому BLAKE2 – ваш выбор, если вам требуется подсчитывать хеши для солидного объема данных. Есть две разновидности BLAKE2: BLAKE2b и BLAKE2s. BLAKE2b предназначен для 64-битных платформ. BLAKE2s разработан для платформ от 8 до 32 бит.

Мы познакомились с безопасными хеш-функциями и узнали, как выбирать между ними. Теперь пора узнать в лицо небезопасные, чтобы избегать их.

2.4.2 Небезопасные хеш-функции

Хеш-функции множества `algorithms_guaranteed` пользуются популярностью и отличаются кросс-платформенностью. Но это не значит, что все они безопасны для криптографических целей. Небезопасные хеш-функции оставлены в Python для обеспечения обратной совместимости. Знать о них стоит, потому что они могут повстречаться вам в устаревших системах. Небезопасные функции среди `algorithms_guaranteed` следующие:

- MD5;
- SHA-1.

MD5

MD5 – устаревшая 128-битная хеш-функция родом из начала 90-х. Это самая широко используемая хеш-функция всех времен и народов. Увы, она до сих пор в ходу, несмотря на то что исследователи продемонстрировали коллизии в ней еще в 2004-м. В наше время криптоаналитикам нужно менее часа, чтобы создать коллизию MD5-хешей на домашнем компьютере.

SHA-1

SHA-1 – устаревшая 160-битная хеш-функция, разработанная NSA в середине 90-х. Как и MD5, эта функция была некогда популярна, но она больше не считается безопасной. Google в сотрудничестве с Центром математики и информатики (Centrum Wiskunde & Informatica), научно-исследовательским институтом, расположенным в Нидерландах, сообщили о первых коллизиях в ней в 2017 году. Говоря языком терминов, они лишили эту функцию сильного сопротивления поиску коллизий. Слабое сопротивление по-прежнему в строю.

Многие разработчики знакомы с SHA-1 по системам контроля версий Git и Mercurial. Там хеши SHA-1 используются для проверки целостности коммитов и их идентификации. Линус Торвальдс, создатель Git, в 2007 году на Google Tech Talk сказал: «Применение SHA-1, во всяком случае в Git, не для безопасности вовсе. Это лишь способ наведения порядка».

> **ВНИМАНИЕ!** MD5 либо SHA-1 ни за что не должны использоваться для целей безопасности при создании новых систем. Любой устаревший сервис, использующий эти функции, должен быть переписан с использованием безопасных альтернатив. Эти функции были некогда популярны, но сейчас популярной и безопасной является SHA-256. Устаревшие функции быстрые, но BLAKE2 еще быстрее и безопаснее.

Итак, вспомним, как стоит выбирать криптографическую хеш-функцию.

- Для большинства задач подходит SHA-256.
- Для обеспечения высокой безопасности подходит SHA3-256, но за это придется заплатить не настолько широкой поддержкой.
- Для объемных сообщений подходит BLAKE2.
- Ни за что не используйте MD5 либо SHA1 для целей безопасности.

В этом разделе вы узнали, как выбрать безопасную криптографическую хеш-функцию. Давайте применим эти знания на практике.

2.5 Криптографическое хеширование в Python

Модуль hashlib содержит в себе именованные конструкторы для каждой хеш-функции из `hashlib.algorithms_guaranteed`. Также все хеш-функции доступны через универсальный конструктор `new`. В качестве аргумента он принимает любую строку из `algorithms_guaranteed`. Именованные конструкторы быстрее, чем универсальный, поэтому стоит использовать их. Код ниже показывает, как создать экземпляр SHA-256 с помощью обоих конструкторов:

```
import hashlib

named = hashlib.sha256()
generic = hashlib.new('sha256')
```

Именованный конструктор

Универсальный конструктор

Экземпляру хеш-функции может быть задано сообщение с самого начала. Код ниже изначально помещает сообщение в экземпляр функции SHA-256. В отличие от встроенной функции `hash`, функции модуля `hashlib` требуют, чтобы сообщение было байтовой строкой:

```
>>> from hashlib import sha256
>>>
>>> message = b'message'
>>> hash_function = sha256(message)
```

Независимо от способа создания, любой экземпляр функции имеет идентичный внешний интерфейс (API). Открытые методы экземпляра SHA-256 не отличаются от открытых методов экземпляра MD5. Методы `digest` и `hexdigest` возвращают хеш как байтовую строку в первом случае и как шестнадцатеричный текст во втором:

```
>>> hash_function.digest()
b'\xabS\n\x13\xe4Y\x14\x98+y\xf9\xb7\xe3\xfb\xa9\x94\xcf\xd1\xf3\xfb"\xf7\x
1c\xea\x1a\xfb\xf0+F\x0cm\x1d'
>>>
>>> hash_function.hexdigest()
'ab530a13e45914982b79f9b7e3fba994cfd1f3fb22f71cea1afbf02b460c6d1d'
```

Хеш в виде байтовой строки

Хеш в виде обыкновенной строки

Следующий пример демонстрирует коллизию MD5-хешей. Для этого используется метод `digest`. У обоих сообщений различаются лишь несколько символов (выделены полужирным):

```
>>> from hashlib import md5
>>>
>>> x = bytearray.fromhex(
...
'd131dd02c5e6eec4693d9a0698aff95c2fcab5812712467eab4004583eb8fb7f8955ad340609
f4b30283e488832571415a085125e8f7cdc99fd91dbdf280373c5bd8823e3156348f5bae6da
cd436c919c6dd53e2b487da03fd02396306d248cda0e99f33420f577ee8ce54b67080a80d1e
```