kaspersky.academy

Лекция 7.

Хэш-функции

Онлайн-курс по математике в информационной безопасности

Лекция 7. Хэш-функции

Привет!

На вашем компьютере лежит файл размером 3,14 гигабайта. И допустим, вам нужно доказать, что этот файл у вас действительно есть. Но не отправлять же столько данных через wi-fi с плохим сигналом! К тому же вы не хотите выдать ваш драгоценный файл, и отправлять его целиком – не вариант.

Вы берете виртуальную мясорубку и отправляете в нее этот длинный-предлинный файл. И одним легким движением руки этот файл превращается в строку фиксированной длины, состоящую из цифр и букв. Такое превращение «из файла в строку» мы будем называть **хэшированием**, а hash означает «превращать в фарш», «перемешивать». Итак, главный герой сегодняшней лекции – **хэш-функции.**

А раз уж мы говорим про хэширование, то нам нужно будет определить ключ хэширования, а заодно подумать о сложностях, которые возникнут при хэшировании. Сложности состоят в появлении коллизий первого и второго рода. Но обо всем по порядку.

Наш план на эту лекцию:

Начинаем!

Что такое хэширование?

Итак, мы будем превращать в фарш данные, поступившие на вход хэш-функции. Несколько определений:

Хэш-функция – это преобразование входных данных в битовую последовательность фиксированной длины

Хэширование – не равно «шифрование». Хэшировать – значит применять хэш-функцию.

Ключ хэширования – данные, которые отправляются на вход хэш-функции. Таким образом, «ключ хэширования» не равно «ключ шифрования».

Прокрутить данные через мясорубку легко, а собрать из фарша стейк невозможно. А это значит, что **хэш-функция будет однонаправленной и необратимой.**

Зачем нужны хэш-функции?

Представьте, что вы регистрируетесь на некотором сайте и придумываете **секретный пароль.** Думали вы думали, а владелец сайта взял и записал ваш хитроумный пароль в свою базу данных в открытом виде. А заодно и пароли всех других пользователей. И теперь, если вдруг злоумышленник доберется до базы данных, **он скомпрометирует вообще всех пользователей!** Проще говоря, он сможет входить на сайт под чьим угодно логином. С этой проблемой нужно что-то делать.

С одной стороны, владелец сайта хочет каждый раз проверять, что пользователь вводит свой пароль правильно, а с другой – записывать ваш пароль в базу данных открытым текстом – просто безумие.

Поэтому в базу данных владелец сайта запишет **не пароли, а хэши от них.** То есть в базе данных хранится хэш от вашего пароля, и, когда вы вводите пароль на сайте, от него считается хэш-сумма и сравнивается с той, что хранится в базе данных. И ваш суперклассный пароль, состоящий и из букв, и из символов (а еще вы добавили несколько нижних подчеркиваний и служебных знаков), даже не будет передаваться по сети, и знать его будете только вы.

№	nickname	hash
1	Obi-Wan	213f
2	toastmaster	a54b
3	sudo-make-me-sandwich	87c4

Таблица 1. Примеры логинов пользователей с хэшами вместо паролей

Одна простая идея для хэш-функции

Приведем вам один маленький пример для хэш-функций. Сразу скажем: идея совсем примитивная и в реальности никто ею пользоваться не будет, но для понимания того, что происходит, этот пример сгодится.

Деление с остатком

При делении с остатком ключу k ставится в соответствие остаток от деления k на некоторое число m.

$$h(k) = k \bmod m$$

Допустим, $m = 5$. Остатки от деления на 5 могут быть такими: 0, 1, 2, 3, 4. Тогда любое число, поданное на вход хэш-функции, отобразится во множество чисел $\{0, 1, 2, 3, 4\}$.

Представьте, что 100 человек нужно разделить на 5 команд. У каждого человека есть свой номер. Тогда достаточно взять хэш от каждого номера, и человек узнает, в какую команду он попал. В случае деления с остатком количество хэш-значений равно числу m.

Применение хэш-функций

Для хэш-функций нашлось много способов применения. Мы рассмотрим самые популярные.

Во-первых, с помощью хэш-функций можно проверять файлы на подлинность. Представьте: вы – программист. И вы долго работали над вашим гениальным приложением, которое облегчит жизнь любому пользователю, установившему его на свой компьютер. Вы публикуете это чудо-приложение на веб-сайте, чтобы каждый человек мог скачать его себе.

Но все не так просто. У вас есть враги, которые хотят помешать вам.

Они подменяют ваш файл своим, и теперь пользователи думают, что скачивают ваш файл, но на самом деле скачивают файл злоумышленника.

Чтобы защитить ваш бесценный файл и всех пользователей, которые будут его запускать, вы решаете взять хэш-функцию от этого файла и опубликовать ее.

Теперь каждый, кто скачивает файл, может взять хэш от него и сравнить с тем, который лежит на веб-сайте.

А мы будем считать, что разным файлам соответствуют разные хэш-функции. Хэш будет своеобразным отпечатком пальца файла – уникальной строчкой, которую можно получить, если взять этот файл в качестве ключа.

Во-вторых, хэш-функции отлично сработают и для проверки целостности файла. Скачиваете вы файл с веб-сайта – а какая-то часть информации потерялась при передаче. И на своем компьютере вы можете проверить, действительно ли файл скачался правильно, или же вам придется выпить еще три чашки чая в ожидании нового скачивания... с хорошими файлами мы разобрались. А что делать с плохими? Например, будут ли у всех вредоносных файлов уникальные хэши?

Да, будут!

В-третьих, мы можем искать вредоносные файлы по хэшам. Допустим, скачиваете вы какой-то файл и не знаете, хороший он или плохой. А что если

это шифровальщик, который навечно зашифрует все ваши важные файлы на компьютере?

Смело берите хэш от этого файла и закидывайте на **VirusTotal** – веб-сайт, который сверит хэш вашего файла с хэшами известных вредоносных файлов и даст вам ответ – запускать вам этот файл или лучше сразу перенести в Корзину.

Базы данных с хэшами хранят еще и антивирусы. Представьте, сколько бы памяти занимали антивирусные программы, если бы им приходилось таскать за собой все вредоносные файлы, которые им известны?

А так – записали им в базы данных хэши вредоносных файлов, и теперь можно по этой базе данных сверять хэши всех файлов на вашем компьютере. Конечно, антивирусы еще и не такое умеют, но хэши они используют во всю.

Коллизии I и II рода на примере Арьи Старк

Как вы видите, бывают случаи, когда хэши от разных чисел совпадают. Например, $5 \bmod 5 = 0$ и $10 \bmod 5 = 0$. Такое явление называется **коллизией**.

Мы начнем с рассмотрения **коллизий первого рода.**

Пусть есть файл A и хэш от него – $h(A)$. Коллизия первого рода произойдет тогда, когда мы сможем подобрать какой-то другой файл B, да еще такой, что:

$$h(A) = h(B)$$

Вернемся в Вестерос.

Тирион и **Серсея** договорились о встрече. Серсея ждет Тириона и узнавать его будет по лицу (условно договоримся, что лицо Тириона – это хэш от Тириона).

Арья хочет незаконно пробраться в Красный Замок и тоже поболтать с Серсеей. Но Арью никто не пустит,

Рисунок 1. Арья хочет пробраться в Красный замок

потому что хэш от Арьи (лицо Арьи) не такой же, как хэш от Тириона (лицо Тириона).

Но Арья умеет менять лица. Она будет выглядеть как Тирион, она возьмет нужную маску, так чтобы хэш от нее совпал с хэшем от Тириона.

Рисунок 2. Арья пробирается в замок с лицом Тириона.

Шалость удалась – **Арье удалось обмануть стражу и пробраться в замок!** Что же будет с файлами?

Помните, вы выложили свой гениальный код на веб-сайт? А что если ваш враг подберет файл, хэш от которого будет равен хэшу вашего файла – тому хэшу, который вы официально опубликовали на веб-сайте?

В этом случае злоумышленник всех обманет, и пользователи будут запускать его файл вместо вашего.

А теперь мы идем к **коллизиям второго рода.**

Итак, если можно **с самого начала** подобрать два сообщения, хэши которых равны, возникает коллизия второго рода:

$$h(A) = h(B)$$

Что это значит?

Это значит, что **Арья и Тирион вступили в сговор с самого начала,** и оба знают, что у Арьи уже есть такое лицо, которое совпадает с лицом Тириона.

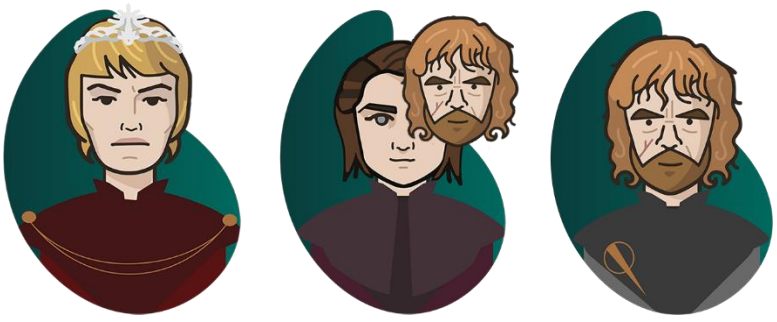

Рисунок 3. Арья и Тирион с самого начала вступили в сговор.

Тогда Тирион пишет Серсее, она ждет его в гости и предупреждает об этом охрану. А Тирион отправляет вместо себя Арью. И охрана ее пропускает. Занавес!

Давайте снова вернемся к файлам.

Как вы, наверное, знаете, производители операционных систем пишут драйверы для них. Но драйверы могут писать и другие производители. А так как драйвер – очень серьезная программа, которая может нанести большой вред компьютеру, драйверы других производителей должны быть одобрены производителем операционной системы.

Но что делают злоумышленники?

Они пишут два файла, один из которых – хороший драйвер, а другой – плохой. **Но хэши у них одинаковые.**

Они приносят хороший драйвер и просят внести его хэш в базу данных «хороших» драйверов. И производитель операционной системы после проверки вносит этот хэш в свою базу данных.

Но злоумышленник не будет распространять хороший файл, он будет распространять плохой. И операционная система разрешит его запустить, потому что она знает, что его хэш помечен как «хороший».

Итого: какой должна быть хорошая хэш-функция?

Во-первых, она должна быть необратимой. Во-вторых, стойкой к коллизиям первого и второго рода.

Весь смысл криптографических хэш-функций в том, чтобы задавать однозначную пару «объект – хэш». А если подмена возможна, да еще и сделать ее легко, то зачем нам такая хэш-функция?

Конечно, коллизии будут случаться во всех хэш-функциях, но подбирать их должно быть ооочень нелегко! Только брутфорс-алгоритмами. Теперь идем решать задачи! :)