Математические аспекты разработки и анализа функций хэширования

Выполнил Емельянов Виталий Дмитриевич
Научный руководитель - доцент каф. Высшая математика
Филимоненкова Надежда Викторовна

Неделя науки Физмех СПбПУ

2022 г.

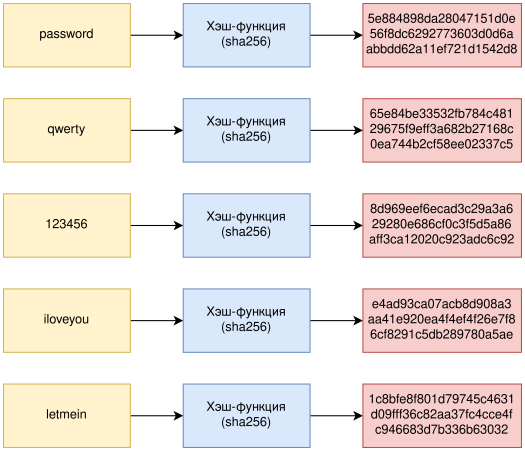
Цели работы

1. Изучить:
   1. понятие хэш-функции с точки зрения информатики и математики
   2. способ проектирования хэш-функций
   3. методы тестирования хэш-функций
2. Реализовать собственное тестирование двух существующих хэш-функций и проанализировать результаты

Что такое хэш-функция?

Применение

Предположим, мы хотим авторизовывать пользователя по паролю. Но хранить сам пароль мы не хотим из соображений безопасности. Тогда мы заменим пароль в нашей базе данных на значение некоторой функции $h(k) : U \to M$, где $k \in U$ - входная последовательность бит.



Коллизии

Каждое совпадение хэш-значения при разных входных аргументах называется коллизией. В идеальной ситуации нам бы хотелось, чтобы коллизий вообще не было. Иначе говоря, чтобы функция была инъективна на всем множестве входных значений U, то есть

$$\forall a, b \in U : h(a) = h(b) \Rightarrow a = b$$

но это невозможно, поскольку множество U - бесконечно, а вот M - конечно.

Устойчивость к коллизиям

В данном случае формулируется требование о «сложном» поиске коллизий, то есть

$$\forall R \Rightarrow \Pr\left[R(h(k)) \in h^{-1}(h(k))\right] < 1/p(n)$$

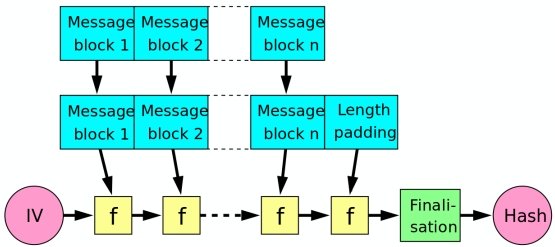
при достаточно большом n, где $\Pr$ - вероятность истинности, R - функция, вычисляемая за полиномиальное от n время, $p(n)$ - любой полином, $n = \log_2(|M|)$ - количество бит в значении хэш-функции, $h^{-1}(h(k))$ - полный прообраз значения $h(k)$.

Проще говоря: вычисление прообраза случайного значения хэш-функции является задачей, не вычислимой за полиномиальное от n время.

Как спроектировать хэш-функцию?

Структура Меркла - Дамгора

Давайте разобьем нашу входную последовательность на ряд подпоследовательностей b_n одной длины и создадим функцию двух аргументов вида $f_n(f_{n-1}(...), b_n)$ и дополнительно определим значение f_0. Значением функции $h(k)$ будем считать f_n, у которого n - наибольший. Эта функция f_n называется функцией сжатия.



Как спроектировать хэш-функцию?

«Устойчивые» функции

При решении подобных задач обычно полагаются на такие математические задачи, как

- Нахождение квадратичного вычета
- Факторизация целых чисел
- Дискретное логарифмирование

Эти задачи считаются неразрешимыми за полиномиальное время. И хороший криптограф пользуется этим и пытается разработать такую функцию, что для ее взлома (то есть, в нашем случае, для нахождения одного из прообразов хэш-значения), злоумышленнику потребуется решить одну из вышеуказанных задач.

Рассмотрим такую функцию, которая будет умножать простые числа, поэтому чтобы ее обратить, потребуется решать задачу по факторизации.
Давайте разработаем нашу функцию свертки, пусть длина нашего блока равна m. Тогда пусть наша хэш-функция будет умножать простые числа в степени бит блока. Таким образом, например, блок 00101100 будет преобразован в значение $2^0 \times 3^0 \times 5^1 \times 7^0 \times 11^1 \times 13^1 \times 17^0 \times 19^0$.
Или, если это писать по-другому, то $\prod_{i=1}^{m} p_i^{b_{ni}}$, где b_{ni} - iй бит блока номер n
Домножим это значение на значение функции свертки от предыдущего блока в квадрате, чтобы учесть его и возьмем по модулю n.

Very Smooth Hash

$$x_{j+1} = x_j^2 \times \Pi_{i=1}^{m} p_i^{k_{j\cdot m+i}} \mod n, x_0 = 1$$

k_i - iй бит аргумента, т.е. $k_{j\cdot m+i}$ - iй бит блока номер m

n - простое число, обозначающее ограничение максимального значения хэш-функции

m - длина блока - наибольшее целое число, такое, что $\Pi_{i=1}^{m} p_i < n$

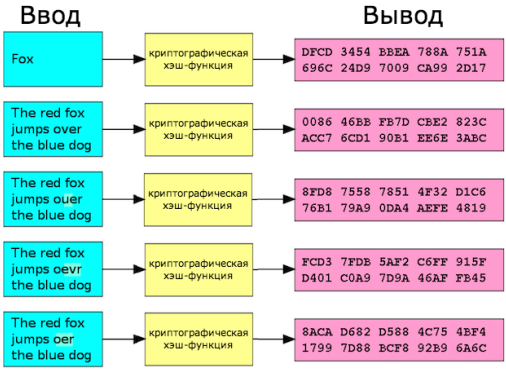
p_i - последовательность простых чисел

Тогда если $\mathcal{L}$ - номер последнего блока, то $h(k) = x_{\mathcal{L}}$

Этот алгоритм называется функцией VSH.

Изначально при восстановлении данных у нас существует неопределенность (или энтропия). Если эта энтропия мала, то это значит, что у злоумышленника будет очень сильно сокращено пространство для перебора входных значений.
По сути, для трудной обратимости необходима максимизация этой самой энтропии, то есть максимальная случайность нашей итоговой последовательности. Для проверки этого существует набор тестов NIST.

Второе условие, которое мы можем сформулировать из требования к устойчивости к коллизиям - «лавинный эффект». Это означает, что малое изменение входных данных приводит к значительному изменению выходных. Это необходимо для того, чтобы атакующий не мог по соседним к искомым входным данным значениям определить, что разница между этими данными небольшая и не мог сократить неопределенность.



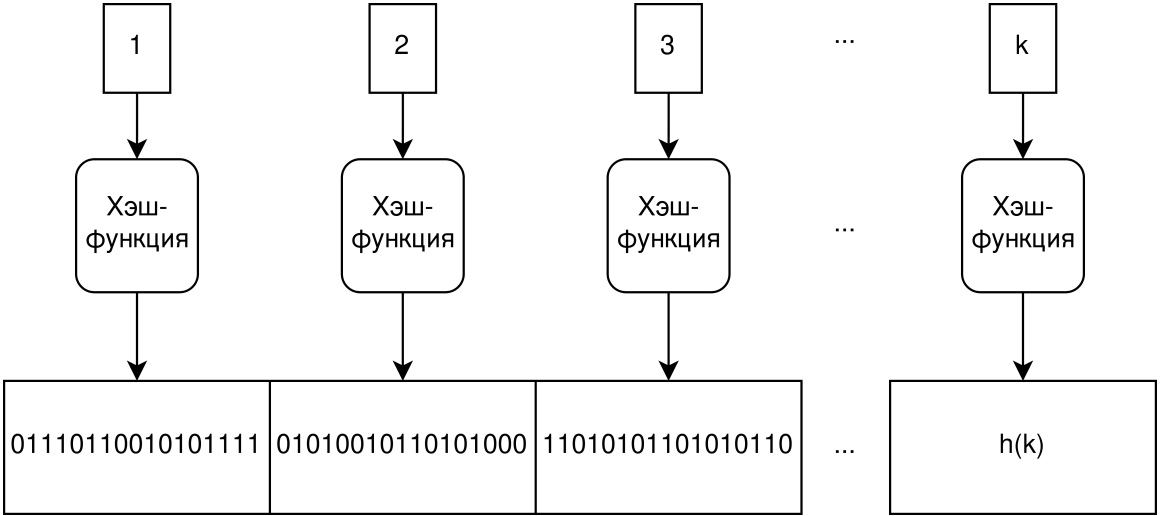
Тестирование хэш-функции

Для тестирования хэш-функций была применена следующая методика. На вход хэш-функций поступают строки в формате Unicode, в каждой из которых записано число. Числа в этих строках идут последовательно. Результаты хэш-функций конкатенируются (то есть записываются последовательно). Итоговая последовательность проверяется NIST-тестированием.

Иначе говоря, проверяется последовательность $h(1) \cdot h(2) \cdot h(3) \cdot \ldots \cdot h(k)$, где $\cdot$ - конкатенация двоичных последовательностей.

Тестирование проводилось самостоятельно с использованием реализации тестов NIST в среде Jupyter Lab.

Тестирование хэш-функции



Анализ результатов тестирования

Результаты тестирования двух хэш-функций - SHA256 и MD5

Тест NIST	SHA256	MD5
Частотный побитовый тест	OK	OK
Частотный блочный тест	OK	OK
Тест на последовательность одинаковых битов	OK	OK
Тест рангов бинарных матриц	OK	OK
Спектральный анализ	OK	OK
Тест на совпадение неперекрывающихся шаблонов	OK	OK
Тест на совпадение перекрывающихся шаблонов	OK	OK
Универсальный статистический тест Маурера	OK	OK
Тест на линейную сложность	OK	OK
Тест на периодичность	OK	OK
Тест приблизительной энтропии	OK	Нарушение

Тест приблизительной энтропии измеряет энтропию по формуле $-\sum_{i=1}^{N} p_i \log p_i$ для блоков длиной m и для блоков длиной $m+1$. Результат - разность этих двух результатов. Если значение мало, то бит можно угадать по предыдущим m блокам, поскольку он вносит малую неопределенность в последовательность.

Анализ хэш-функции

Тест приблизительной энтропии. Результаты

Длина блока m в тесте	Результат для md5
1	OK
2	OK
3	OK
4	OK
5	OK
6	OK
7	Нарушение
8	Нарушение
9	OK
10	Нарушение

Анализ хэш-функции

MD5

```
d131dd02c5e6eec4693d9a0698aff95c2fcab58712467eab4004583eb8fb7f89
55ad340609f4b30283e488832571415a085125e8f7cdc99fd91dbdf280373c5b
d8823e3156348f5bae6dacd436c919c6dd53e2b487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080a80d1ec69821bcb6a8839396f9652b6ff72a70
```

and

```
d131dd02c5e6eec4693d9a0698aff95c2fcab50712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325f1415a085125e8f7cdc99fd91dbd7280373c5b
d8823e3156348f5bae6dacd436c919c6dd53e23487da03fd02396306d248cda0
e99f33420f577ee8ce54b6708080d1ec69821bcb6a8839396f965ab6ff72a70
```

В марте 2005го года была опубликована статья, в которой описывался алгоритм, позволяющий найти две одинаковые последовательности длиной 128 бит с одинаковым значением хэш-функции MD5.

Источники

1. https://bit.nmu.org.ua/ua/student/metod/cryptology/лекция17.pdf - описание криптографической хэш-функции
2. https://cryptography.ru/ref/функция_односторонняя/ - описание односторонней функции и трудной обратимости
3. https://eprint.iacr.org/2005/193 - описание функции VSH
4. https://vital.lib.tsu.ru/vital/access/services/Download/vital:11184/SOURCE01
5. https://www.mscs.dal.ca/~selinger/md5collision/ - коллизии MD5