Алгоритм хэширования Keccak и лавинный эффект.

Батарин Егор

Аннотация

Статья посвящена обзору самому передовому на данный день алгоритму хеширования - Keccak. Рассмотрены история его создания, определение хэш-функции, основные свойства криптографических хэш-функций, общая структура алгоритма Keccak, реализация функции перестановок и описание проекта по исследованию лавинного эффекта на Keccak.

1 История создания алгоритма

В 2012 году сообщество NIST провело конкурс алгоритмов SHA-3, после того, как были проведены успешные атаки на предыдущие алгоритмы семейства SHA. На конкурс были приняты 51 алгоритмов, в результате естественного отбора которых в качестве нового стандарта SHA был принят алгоритм Keccak.

2 Что такое хэш-функция?

Хэш-функция - это отображение, которое ставит в соответствие строке произвольной длины строку фиксированной длины. Как правило, мощность отображаемого множества больше мощности множества значений хэш-функции, а значит она не может быть инъективной. Пара сообщений, нарушающих инъективность функции, называется коллизией.

Сферы применения хэш-функций различны:

- Построение уникального идентификатора для данного набора данных
- Для вычисления контрольных сумм и обнаружения ошибок
- При сохранении пароля в виде хэш-кода
- При выработке электронной подписи на хэш-код

Последние два применения взяты из криптографии. Не любые хэш-функции годятся для применения в этих случаях. Для этого нужно использовать специальные хэш-функции, называемые криптографическими, которые обеспечивают повышенную безопасность системы.

3 Свойства криптографических хэщ-функций

Чтобы хэш-функция была годна для защиты информации, она должна обладать следующими свойствами:

-Сопротивление поиску первого прообраза: для данного хэш-кода трудно найти исходное сообщение, которое было отображено данной хэш-функцией в данный хэш-код. Иными словами, для данной хэш-функции трудно построить функцию, обратную к ней.

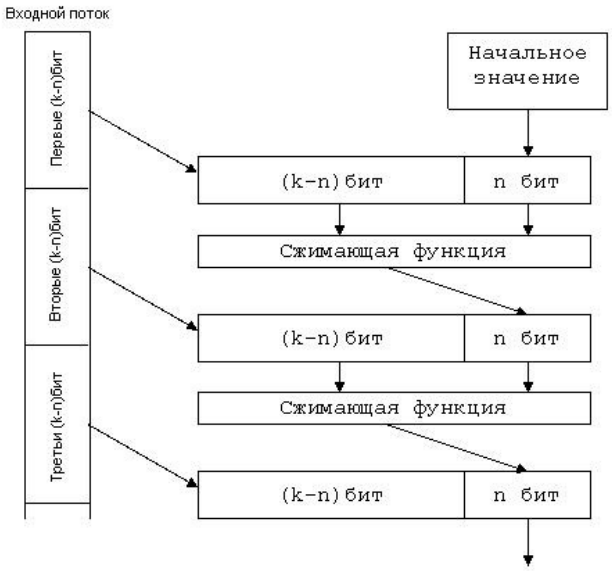
-Сопротивление поиску второго прообраза: для данного сообщения трудно найти другое сообщение, отображаемое хэш-функцией в тот же хэш-код.

-Стойкость к коллизиям: нет эффективного полиномиального алгоритма, который находит коллизии.

-Лавинный эффект: малое изменение входных значений хэш-функции сильно меняет выходное значение.

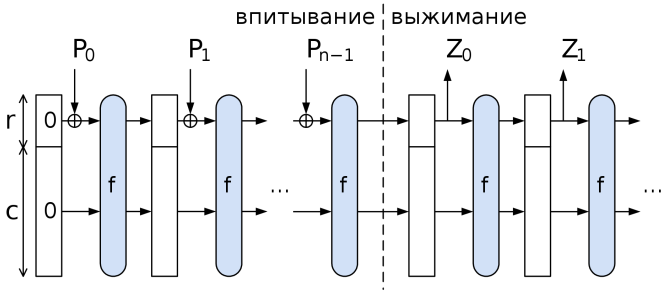
Как правило, криптографические хэш-функции строятся по итеративной последовательной схеме.

Такая схема позволяет достичь лавинного эффекта, поскольку каждый бит выходного потока зависит от всего входного потока данных. В основе распространенной схемы лежит разделение входного сообщения на множество блоков, каждый из которых обратывается сжимающей функцией. Затем полученное значение используется для последующих итераций. Для того, чтобы длина входного сообщения была кратна длине блока, его удлинняют дополнительными битами до нужной длины. Похожие идеи используются в алгоритме Keccak. Такая констукция называется губкой.



4 Губка

Вычисление хэш-кода в констукциях по типу губки происходит в несколько стадий.



1) Исходное сообщение M дополняется до строки P так, чтобы длина последнией была кратна r. Для этого используется функция дополнения.

2) Строка P делится на строки P_0,...,P_{n-1}.

3) Происходит этап "впитывания":

3.1) Каждый блок P_i дополняется нулями до строки длины b бит - верхние входы над вертикальными стрелочками на рисунке

3.2) Генерируется начальная строка состояния S длины $b = r + c$, заполненная нулями - самый левый белый прямоугольник на рисунке.

3.3) Она суммируется по модулю 2 вместе со строкой P_0, а результат обрататывается функцией перестановок f.

3.4) Полученная в результате действия f строка длины b суммируется по модулю 2 с P_1 и все последующие операции повторяются итеративно.

4) Происходит этап "отжимания": на этом этапе суммирование по модулю 2 прекращается и к строке состояния применяется функция перестановок. К конечному результату Z каждый раз добавляются первые r бит строки состояние. Происходит это ддо того момента, пока длина Z будет не меньше d. Когда это достигнуто, лишняя часть обрезается и в конце концов получается строка Z длины d.

5 Функция перестановок f

5.1 Общая структура

Как мы видели выше, строка состояния S претерпевает изменения под действием функции перестановок f в результате "впитывания"и "отжимания". Авторы представляют эту строку как массив $5 \times 5 \times 64$, стало быть $A[i][j][k]$ - это $(5i + j) \times 64 + k$ бит строки S. В общей реализации SHA-3 вместо 64 стоит $w = 2^l$, при этом вычисления, проходящие во время определения результата функции перестановок, проходит в несколько раундов, число которых равно $12 + 2l$ алгоритма Keccak положено $l = 6$, так что для него число раундов равно 24. В каждый из этих раундов проходит вычисление функций θ, ρ, π, χ, ι. Опишем подробно сущности этих пяти шагов и договоримся на каждом шаге обозначать входной массив за A, а выходной за A'. mod означает сложение по модулю 2. i_r - номер раунда.

5.2 Шаг θ

Для всех i и k, таких, что $0 \leqslant i < 5$, $0 \leqslant k < w$, положим
$C(i, k) = A[i, 0, k] \oplus A[i, 1, k] \oplus A[i, 2, k] \oplus A[i, 3, k] \oplus A[i, 4, k]$
$D(i, k) = C[(i - 1) \bmod 5, k] \oplus C[(i + 1) \bmod 5, (k - 1) \bmod w]$
Для всех (i, j, k), таких, что $0 \leqslant i < 5$, $0 \leqslant j < 5$, $0 \leqslant k < w$,
$A'[i, j,k] = A[i, j, k] \oplus D[i, k]$

5.3 Шаг ρ

Для всех k, таких, что $0 \leqslant k < w$, $A'[0,0,k] = A[0,0,k]$
Пусть в начале $(i, j) = (1, 0)$.
Для t от 0 до 23 выполнять:
1. Для всех k, таких, что $0 \leqslant k < w$, $A'[i,j, k] = A[i, j, (k - (t + 1)(t + 2)/2) \bmod w]$
2. $(i, j) = (j, (2i + 3j) \bmod 5)$

5.4 Шаг π

Для всех (i, j, k), таких, что $0 \leqslant i < 5$, $0 \leqslant j < 5$, $0 \leqslant k < w$
$A'[i,j,k] = A[(i + 3j) \bmod 5, i, k]$

5.5 Шаг χ

Для всех (i, j, k), таких, что $0 \leqslant i < 5$, $0 \leqslant j < 5$,
$A'[i,j,k] = A[i,j,k] \oplus ((A[(i + 1) \bmod 5, j, k] \oplus 1) \cdot A[(i + 2) \bmod 5, j, k])$

5.6 Шаг $\iota(A,i_r)$

Введем дополнительную функцию $rc(t)$. Она вычисляет значения следующим образом:
1. Если $t \bmod 255 = 0$, то возвращается 1
2. Пусть $R = [10000000]$
3. Для i от 1 до $t \mod 255$ выполнять:
   3.1) R = 0 || R
   3.2) $R[0] = R[0] \oplus R[8]$
   3.3) $R[4] = R[4] \oplus R[8]$
   3.4) $R[5] = R[5] \oplus R[8]$
   3.5) $R[6] = R[6] \oplus R[8]$
   3.6) $R = Trunc_8[R]$
4) Возвращается $R[0]$

Теперь описываем сам алгоритм $\iota(A,i_r)$:
1. Для всех (i, j, k), таких, что $0 \leqslant i < 5$, $0 \leqslant j < 5$, $0 \leqslant k < w$ $A'[i,j,k] = A[i,j,k]$
2. Положим RC — массив длины w, заполненный нулями.
3. Для i от 0 до l: $RC[2^i - 1] = rc(i + 7i_r)$
4. Для всех k, таких, что $0 \leqslant k < w$, $A'[0,0,k] = A'[0,0,k] \oplus RC[k]$

5.7 Алгоритм перестановок

Теперь можно писать структуру всего алогоритма перестановок:
1. Переводим строку S в массив A
2. Для i_r от $12 + 2l - n_r$ до $12 + 2l - 1$ применяем $A' = \iota(\chi(\pi(\rho(\theta(A)))), i_r)$
3. Переводим конечный массив A' в строку S'

6 Криптоанализ

На данный момент алгоритм Keccak является самым передомым и надежным среди всех существующих алгоритмов. Тем не менее, существуют известные попытки взлома алгоритма. Рассмотрим некоторые из них.

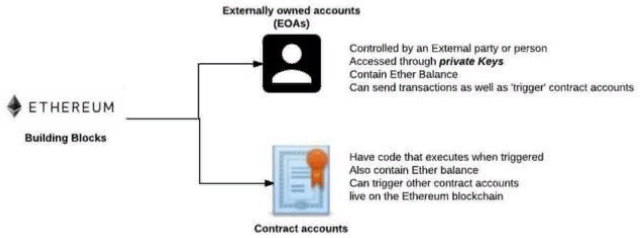
В работе [7] приводится алгоритм, позволяющий найти коллизии в 384-битной версии алгоритма Keccak с 4 раундами за практическое время $2^{59.65}$. Хотя данный алгоритм неприменим на практике из-за огромного вычислительного времени (иначе пришлось бы разрабатывать SHA-4), он дает существенные продвижения по эффективным аттакам на Keccak, демонстрируя большой выигрыш с предыдущими попытками с практическим временем 2^{147}.

Работа [8] представляет алгоритм поиска коллизий уже 6 раундовом алгоритме Keccak с временем $2^{123.5}$, причем с использованием квантовых вычислений достигается ускорение вплоть до $2^{67.35}$.

Аттака на прообраз описана в работе [9]. Здесь аттаки проводятся на алгоритме Keccak 384/512 с использованием линейных структур.

7 Применение

Один из самых ярких примеров использования алгоритма Keccak - криптовалюта Ethereum. Здесь этот алгоритм используется для хэширования Ethereum адресов.



Сами адреса в Ethereum разделяются на два типа:

1) External owned accounts (учетные записи, принадлежащие внешним пользователям, EOA): контролируются закрытыми ключами.

2) Contract accounts (учетные записи смарт-контрактов, CA): самоуправляемы своим своим кодом.

В Ethereum адрес имеет размер в 20 байт . Он соответствует последним 20 байтам хэша Keccak-256 открытого ключа. Сама процедура создания адреса устроена следующим образом:

1) Создается открытый ключ из закрытого с помощью ECDSA

2) Применяем Keccak к открытому ключу

3) Берем последние 20 байт из хэша - это адрес

8 Список литературы

1. FIPS PUB 202
2. SHA-3 Derived Functions
3. Статья в Википедии про SHA-3
4. Пост в Habr про Keccak
5. Статья в Википедии про функцию губки про хэш-функцию
6. Статья в Википедии про функцию губки
7. Finding Collisions against 4-Round SHA-3-384 in Practical Time
8. Exploring SAT for Cryptanalysis: (Quantum) Collision Attacks against 6-Round SHA-3
9. Improved Preimage Attacks on Round-Reduced Keccak-384/512 via Restricted Linear Structures
10. Учебник по Solidity. Все об адресах / Хабр