### Коллизии хэш-функций

Стойкость современных хэш-функций часто основывается на предположении об их стойкости к нахождению коллизий.

Пусть H: {0,1}^\* -> {0,1}^n функция, переводящая множество двоичных векторов произвольной длины в множество векторов фиксированной длины n. n называется размеров выхода хэш-функции H.

Хэш-функция H называется стойкой к коллизиям второго рода, если вычислительно сложно найти такую пару (x,y) \in {0,1}^\* : x != y, H(x) = H(y).

Очевидно, что даже для "идеальной" хэш-функции (т.е. хэш-функции, на которую не существуют атак лучше перебора) существуют коллизии, из-за разности мощностей множества прообразов и множества образов хэш-функции.

### Атака на основе парадокса дней рождений

Рассмотрим атаку на основе так называемого парадокса дней рождений.

0. S <- пустое множество.  
1. Выбрать произвольный элемент x из {0,1}^\*  
2. Вычислить значение h\_x = H(x)  
3. Проверить, есть ли в множестве S элемент (y, h\_x). Если есть => goto DONE  
4. Добавить элемент (x, h\_x) в множество S  
5. goto 1  
DONE:  
6. Вернуть пару (x, y) как коллизию хэш-функции.

Данная атака на некоторую стойкую к коллизиям хэш-функцию H: {0,1}^\* -> {0,1}^n имеет вычислительную сложность O(2^(n/2)) и требует O(2^(n/2)) памяти.

Объяснение "на пальцах" почему откуда взялся корень.

**NB!** Это только наглядное объяснение, а не доказательство! Доказательство вводится формально на лекции.

При выборе q-го элемента число пар составляет q(q-1) ~ q^2. При этом вероятность коллизии в каждой паре - 1/2^n (совпадение двух случайных векторов). Хотим вероятность коллизии порядка 1, следовательно q^2/2^n ~ 1, q=sqrt(n).

### Атака на основе метода Полларда

Рассмотрим атаку на основе метода Полларда. Основная идея - уменьшить требования к памяти, используя параллельные вычисления.

Пусть имеется m потоков выполнения программы. Пусть Pi : {0,1}^a -> {0,1}^(a+k) - инъективная функция: x |-> x || 0^k, где 0^k - нулевой двоичный вектор длины k. q=b/2 - log\_2(m). H: {0,1}^\* -> {0,1}^b

Отличительной точкой (distinguished point) назовём такой вектор y, у которого первые q бит - нулевые.

Каждый поток инициализируется уникальным начальным значением y\_0. Начальные значения сохранены в памяти.

Основная идея - каждый поток вычисляет последовательно цепочку хэшей, с применением поверх хэша инъективной функции.

y\_i = Pi(H(y\_{i-1})

Номер текущей итерации для каждого потока (i) также сохранён в памяти.

После чего производится проверка, является ли получено значение отличительной точкой. Если является - проверяем, встречали ли мы такую точку ранее. Если нет - добавляем в множество найденных точек и продолжаем алгоритм.

Если данная точка уже встречалась - значит где-то в построенных цепочках встречались 2 различных прообраза. Для найденной отличительной точки имеем 2 номера итерации i и j. Пусть i>j Находим разность между ними d=i-j.

Применяем d итераций к начальной точке y, соответствующей цепочке большей длины. Далее синхронно итерируем цепочки, пока не найдём коллизию.

Сложность нахождения отличительной точки на одном потоке - O(2^{b/2}), на m потоках - O(2^{b/2}/m) Потребление памяти порядка O(m)

0. S <- пустое множество. Общий доступ для всех потоков.  
0.

Каждый поток выполняет следующий алгоритм:

1. y\_i = Pi(H(y\_{i-1})  
2. Если y\_i != 0^q || yr\_i: goto 1  
3. Если (z\_j, j) : z\_j == y\_i, для любого j находится в S: goto FOUND  
4. Добавить (y\_i, i) в S  
5. goto 1  
FOUND:   
6. Пусть i>j. Вычислить d = i-j  
7. Вычислить y = ((Pi\*H)^d)(y\_0), z = z\_0; где (Pi\*H)(u)=Pi(H(u))  
8. Если H(y) = H(z): вернуть (y,z) как коллизию хэш-функции.  
9. (y, z) = (Pi(H(y)), Pi(H(z)))  
10. goto 8

### Выполнение лабораторной

1. Реализовать "усеченные" хэш функции sha-XX, путём взятия первый XX бит функции sha-256. XX = {15-20}
2. Найти коллизию используя парадокс дней рождений для хэш функций sha-XX.
3. Найти коллизию, используя Ро-метод Полладра для хэш функций sha-XX на двух потоках.

Для пунктов 2 и 3 оценить используемую память для каждого метода. Для каждого из пунктов найти не именее 100 (больше - лучше) коллизий для каждой хэш-функциий. Замерить время выполнения. Построить график зависимости среднего времени выполнения от размера выхода хэш-функции и средних затрат памяти памяти от размера выхода хэш-функции для обоих методов нахождения коллизий. Затраты по памяти можно для простоты оценить приблизительно как количество хранимых хэшей, умноженное на размер хэша + размер прочих хранимых данных в байтах.

### Результат работы:

код, коллизии для хэш-функций (список hex значений), графики.

Доп литература:

https://github.com/CryptoCourse/CryptoLectures/blob/master/Lectures/Lecture11.pdf (стр 7-8, 26-29, 43-46)

https://iamaaditya.github.io/2012/07/the-birthday-paradox-proof/

https://crypto.stackexchange.com/questions/52231/what-is-the-best-and-fastest-algorithm-to-generate-a-hash-collision

https://cr.yp.to/hash/collisioncost-20090517.pdf (стр 9-10)